

RESEARCH ARTICLE

Flexible multi-fidelity framework for load estimation of wind farms through graph neural networks and transfer learning

Gregory Duthé¹ , Francisco de N Santos², Imad Abdallah^{1,3}, Wout Weijtjens² ,
Christof Devriendt² and Eleni Chatzi¹ 

¹Institute of Structural Engineering, ETH Zürich, Switzerland

²OWI-Lab, Vrije Universiteit Brussel, Belgium

³RTDT Laboratories AG, Zürich, Switzerland

Corresponding author: Gregory Duthé; Email: duthe@ibk.baug.ethz.ch

Received: 31 January 2024; **Revised:** 04 June 2024; **Accepted:** 20 August 2024

Keywords: Graph neural networks; Population-based structural health monitoring; Predictive models; Transfer learning; Wind farms

Abstract

With global wind energy capacity ramping up, accurately predicting damage equivalent loads (DELs) and fatigue across wind turbine populations is critical, not only for ensuring the longevity of existing wind farms but also for the design of new farms. However, the estimation of such quantities of interests is hampered by the inherent complexity in modeling critical underlying processes, such as the aerodynamic wake interactions between turbines that increase mechanical stress and reduce useful lifetime. While high-fidelity computational fluid dynamics and aeroelastic models can capture these effects, their computational requirements limits real-world usage. Recently, fast machine learning-based surrogates which emulate more complex simulations have emerged as a promising solution. Yet, most surrogates are task-specific and lack flexibility for varying turbine layouts and types. This study explores the use of graph neural networks (GNNs) to create a robust, generalizable flow and DEL prediction platform. By conceptualizing wind turbine populations as graphs, GNNs effectively capture farm layout-dependent relational data, allowing extrapolation to novel configurations. We train a GNN surrogate on a large database of PyWake simulations of random wind farm layouts to learn basic wake physics, then fine-tune the model on limited data for a specific unseen layout simulated in HAWC2Farm for accurate adapted predictions. This transfer learning approach circumvents data scarcity limitations and leverages fundamental physics knowledge from the source low-resolution data. The proposed platform aims to match simulator accuracy, while enabling efficient adaptation to new higher-fidelity domains, providing a flexible blueprint for wake load forecasting across varying farm configurations.

Impact Statement

With wind turbines increasing in size and capacity, accurately predicting wind turbine fatigue has become critical for both wind farm design and operation. Specifically, there is a need for load forecasting given far-field wind inflows for unseen farm layouts. We present a novel graph neural network framework that enables local flow, power and fatigue load prediction for several turbine components (tower, blades) over an entire wind farm. We train a foundational model on a large dataset of diverse low-fidelity simulations, that can be easily transferred and adapted to new farms and turbine types. This approach produces a flexible pre-trained generalist model that understands the underlying physics, requiring little data to fine-tune for prediction at higher-fidelities. Our approach opens up new avenues for wind farm design and control that take into account fatigue.

  This research article was awarded Open Data and Open Materials badges for transparent practices. See the Data Availability Statement for details.

© The Author(s), 2024. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (<http://creativecommons.org/licenses/by/4.0>), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

1. Introduction

As the world transitions toward a more sustainable energy landscape, wind energy has emerged as a pivotal component in meeting renewable targets. Projected to grow to 15% of the global electricity supply by 2030 (IEA, 2023), wind has become an increasingly cost-effective alternative to fossil fuel power plants, thanks to advancements in both wind turbine technology and simulation capabilities. However, optimizing the performance and longevity of wind farms still presents significant challenges, particularly when it comes to accurately predicting load accumulation in wind turbine structures (Veers et al., 2022). The complex aerodynamic interactions between turbines, known as wake effects, can not only lead to sub-optimal energy output but also to increased mechanical stress on turbine components and added structural loads on fatigue-critical locations. Understanding and quantifying wake-induced increases in damage equivalent loads (DELs, also referred to simply as “loads”) is crucial, as these factors can significantly impact the remaining useful lifetime (RUL) of a wind turbine. For newer offshore wind farms this has become an operational concern, as structural reserves have diminished with improved design codes (de N Santos et al., 2022).

Concerns with the structural integrity of wind turbines, specifically offshore, have lead to increasingly sophisticated sensor instrumentation (Barber et al., 2022), with data-driven approaches capitalizing on the greater availability of data (de N Santos et al., 2023b), especially of farm-wide accelerometer installation (de N Santos et al., 2024b). In cases where instrumentation is not possible, and where greater flow resolution or load forecasting is required, Computational Fluid Dynamics (CFD) methods in combination with aeroelastic models have been employed to quantify wake effects and the resulting loads impacting a turbine given a certain inflow condition. However, due to the sheer range of modeling scales required to represent full wind farms, highly accurate methods like Large Eddy Simulations (LESS, Breton et al., 2017, Sood et al. 2022) become overly computationally taxing (Veers et al., 2019). Thus, in practice, wind farm designers adopt simpler approaches, namely aeroelastic models based on the blade element momentum method (Madsen et al., 2020) such as HAWC2 (Larsen and Hansen, 2007) or OpenFAST (OpenFAST, 2016). Computational expense becomes even more critical for the near real-time decision-making tasks that farm operators face, and the increase in demand for accelerated wind farm simulation, with its inevitable trade-off between fidelity in the delivered prediction and reduced computational requirements. One such simulator, which has been gaining popularity within the wind energy community, is PyWake (Pedersen et al., 2019), a Python-based wind farm simulator that has the ability to model both the local flow and the loads in a rapid manner. On the other hand, due to the aforementioned trade-off, an increasing focus has been given to surrogate modelling of higher-fidelity simulators through machine learning tools (Wilson et al., 2017; Schøler et al., 2023) attempting to bridge this gap by providing surrogates capable of both fast and accurate modelling. Nevertheless, the research-community still has not coalesced into a clear methodology able of presenting satisfying results, especially because the majority of surrogates are highly specific to a single task. A robust, fast, and flexible platform for wake-induced DELs on turbines in a wind farm is still to materialize. Such a platform would be particularly valuable, as it could be leveraged not only for wind farm design but also for improved control.

Wakes and their progression through a population of turbines are not restricted to single turbines but arise from the description of the full flow through a wind farm. The propagation of wake and the recovery of the boundary layer takes place over the entire farm, thus favoring a modelling strategy that accounts for this spatial configuration. Capitalizing on the concept of population-based structural health monitoring we propose an approach that considers the turbines within a wind farm as a homogeneous population (Bull et al., 2021) that is interconnected, thus forming a graph (Gosliga et al., 2021). We develop a graph-based surrogate simulation framework, which remains as flexible as any physics-based simulator, yielding comparable accuracy while computing considerably faster. The framework that we present is a blueprint for a versatile tool for flow, power, and DEL prediction on wake-affected turbines. Moreover, through transfer learning, we can flexibly extend our prediction even for turbine types that have not been used in our initial training set (Gardner et al., 2021).

Our approach hinges on the use of Graph Neural Networks (GNNs, Veličković (2023)), which are particularly valuable because, unlike regular neural networks, GNNs are not constrained to a single grid/layout. Regular neural networks, such as Convolutional Neural Networks (CNNs), typically operate on fixed-size inputs arranged in a grid-like structure, like images. In contrast, GNNs are designed to work with data represented as graphs, which consist of nodes (representing entities) and edges (representing relationships between entities). This flexibility is achieved through the message-passing paradigm (Gilmer et al., 2017), where nodes in the graph exchange information with their neighbors iteratively (as detailed in Algorithm 1). During this process, each node aggregates information from its adjacent nodes, updates its state, and passes the updated information to the next layer. This mechanism allows GNNs to capture complex dependencies and interactions within the data. Furthermore, GNNs can encode task-specific non-linearities within the graph architecture (Xu et al., 2020). By incorporating various forms of non-linear transformations during the message-passing steps, GNNs can model complex patterns and relationships tailored to specific tasks, enhancing their adaptability to diverse applications. This trait renders GNNs extraordinarily powerful and flexible, enabling them to handle a wide range of problems from social network analysis to molecular chemistry. We leverage these properties by modelling wind farms as graphs, and train a layout-agnostic GNN surrogate using a dataset of low-fidelity PyWake simulations of randomized wind farm layouts simulated under diverse inflow conditions. The intuition behind this initial training is to produce a foundational model capable of understanding the basic underlying physics. We then fine-tune the pre-trained GNN, transferring it for higher-fidelity modelling of a specific wind farm (Lillgrund) that is comprised of different wind turbines to those used in the PyWake simulations.

This pre-training procedure allows us to bootstrap the training of the higher-resolution fine-tuned Lillgrund model, requiring only a minimal amount of high-fidelity HAWC2Farm simulations. In particular, we use modern fine-tuning techniques to ensure efficient, readily-accessible adaptation of the pre-trained GNN, allowing us to produce accurate models in scenarios where limited data hinders training from scratch. We experiment with this framework and further proceed to offer guidelines on the lowest amount of new data required to obtain a target range of acceptable error when fine-tuning the foundational model. We provide the associated code and pre-trained models at <https://github.com/gduthet/windfarm-gnn>.

The remainder of this article is organized as follows. We first provide an general overview of our methodology in section 2. Then, we cover the relevant literature in terms of wind farm modeling and GNNs as well as approaches for transfer learning and fine-tuning in section 3. In section 4, we describe the two different data sources used in our framework, and in section 5 we detail our GNN architecture and describe the different methods used to perform fine-tuning on the higher fidelity data. Finally, in section 6 we show results and discuss their implications, followed by concluding remarks in section 7.

2. Methodological overview

As stated in the introductory section, this contribution attempts to develop a flexible method for multivariate prediction on wind farms. Specifically, we attempt to estimate the fatigue and aerodynamic loads for random inflow conditions, given a wind farm layout that is unseen during model training, or in other words not used in the training set. We build upon the methodology described in Duthæt et al. (2023b) and in de N Santos et al. (2024a), wherein PyWake aeroelastic simulations are used to train and validate GNNs. In this work, we rely on the speed of PyWake to easily generate load estimations for a wide range of inflow conditions and farm layouts. These are used to train a 'generalist' GNN, i.e. a baseline GNN capable of understanding the fundamental physics of the problem. After the training of the generalist GNN we fine-tune it for a specific farm and turbine configuration (Lillgrund), given a limited dataset of higher-fidelity simulations modelled with HAWC2Farm. In this contribution, in particular, we compare three fine-tuning techniques: full model tuning, decoder-only tuning and LoRA. An overview of our approach can be observed in Figure 1.

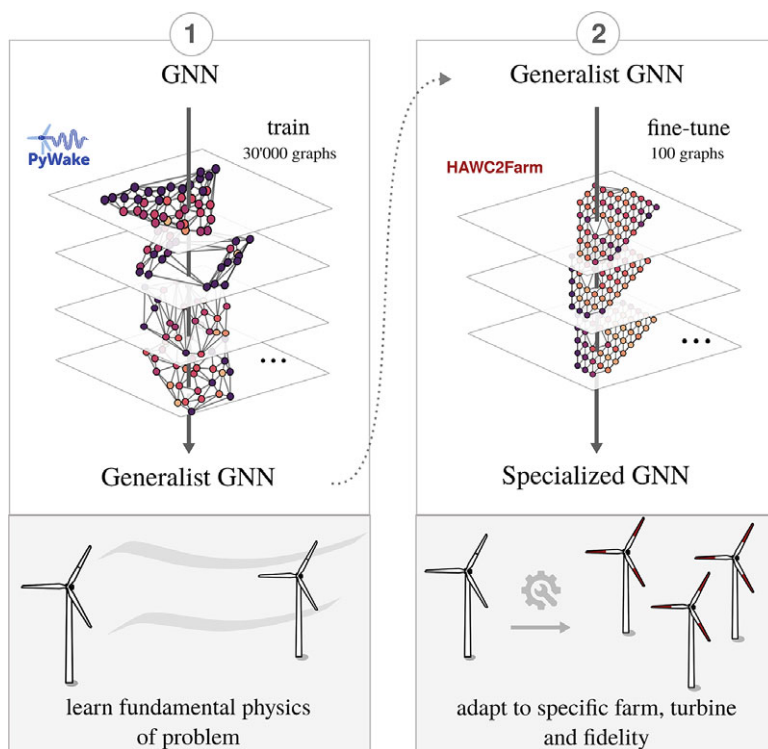


Figure 1. Methodological overview. We first train a generalist GNN on a large dataset of random PyWake simulation graphs, giving it an understanding of the underlying physics. Then, we fine-tune the model for a specific wind farm with a small dataset of higher-fidelity HAWC2Farm simulations.

3. Related work

3.1. Current modelling of wind farms

The operation, layout optimization and control of modern wind farms remain open areas of study as load estimation under real conditions is still difficult. This difficulty lies fundamentally with the complex interactions between wind farms and the highly turbulent, non-stationary and often heterogeneous atmospheric boundary layer, along with the intra-farm turbulent wake flows (Porté-Agel et al., 2020). A further complexity stems from the fact that wind farms are affected by wide-ranging flow conditions and turbulent scales, from the weather meso- and macroscale to the atmospheric boundary layer integral scale and Kolmogorov microscale (Veers et al., 2019). It is the multi-scale nature of the bidirectional influence between turbines and flow that increases the problem's complexity.

There exist various approaches for modelling the interaction of wind turbine and wind farm with flow. Apart from experimental approaches (which we will not approach in this contribution), so-called 'engineering' or analytical methods, and CFD can be used. The first class predicts wake deficits by applying basic physical equations that govern the conservation of flow properties (e.g., mass, momentum and energy) for different regions (near- and far-wake) (Porté-Agel et al., 2020). The flow solver PyWake implements a series of these methods, which are relatively fast. However, these methods are necessarily less accurate than numerical models due to their simplifications. This is particularly troublesome for fatigue load estimation, as these depend on a three-dimensional description of the flow field.

Numeric/CFD models, on the other hand, like Reynolds-Averaged Navier-Stokes (RANS) and LES, give more accurate descriptions of wind farm flow fields, but they are still too computationally expensive for industry employment, although advances made, particularly for LES, might yet open the door to near

real-time LES-based control (Janssens and Meyers, 2023). Therefore, aeroelastic simulations at the farm level necessarily represent a trade-off between precise load prediction and computational cost (Liew et al., 2022). Thus, a clear gap exists between fast and coarse, and slow and fine simulations. This has led to use of surrogate modeling techniques, frequently implemented through machine learning algorithms (Simpson et al., 2021; Mylonas et al., 2021a), for estimating wind farm dynamic responses. The present contribution inserts itself in this constellation, where we attempt to employ machine learning techniques (GNNs) for surrogate modelling of flow solvers. We specifically attempt to bridge the gap between different fidelity-level solvers (PyWake and HAWC2Farm) through multi-fidelity transfer learning.

3.2. GNNs and their applications for physics-based problems

Graph-based deep learning (Battaglia et al., 2018), which is a subset of geometric deep learning (Bronstein et al., 2017), is a type of machine learning that is non-Euclidean, meaning that is not restricted to fixed-size inputs and outputs. It is increasingly being explored for its potential in addressing a variety of problems in physics, as indicated by research in Sanchez-Gonzalez et al. (2018) and expanded upon in Sanchez-Gonzalez et al. (2020). This growing adoption can be attributed to its powerful representational capabilities. Although the development of learning on graphs has been relatively slower due to challenges arising from graph-encoded data, such as differentiability issues caused by graph discreteness or loops, recent years have witnessed significant progress in deep learning on graphs. Building upon foundational works (Scarselli et al., 2008; Micheli, 2009), these advancements have propelled graph-based deep learning to the forefront. As a result, graph-based methods have become integral to the machine learning community, with algorithms developed for deep learning on graphs finding widespread adoption across various domains. For instance, Pfaff et al. (2020) provide an example of GNNs being employed to emulate the operations of conventional mesh-based physics simulators, with an ability to forecast the progression of transient solutions. Our methodology also adopts the Encode-Process-Decode pipeline introduced in the aforementioned works, as it has proven to be effective at learning underlying physical processes in multiple different domains. One such domain, is the reconstruction of flows around airfoils in high Reynolds turbulent inflow conditions (Duthé et al., 2023a). This setup has also seen success when combined with physics-informed objective functions (Hernández et al., 2022). Moreover, GNNs have proven to be adept at approximating the solutions to PDEs (Li et al., 2020b) and even performing medium-range weather forecasting at a level that is on par or superior to the currently-used computational methods (Lam et al., 2023).

3.3. GNNs for wind farms

Despite still being a nascent subfield of machine learning, GNNs have attracted a significant amount of attention in the domain of wind energy applications, with their usage primarily concentrated on power (Bentsen et al., 2022; Yu et al., 2020; Park and Park, 2019) and wake loss predictions (Bleeg, 2020; Li et al., 2022; Levick et al., 2022). Bentsen et al. (2022) employ graph attention networks (GAT, Veličković et al. (2017)) to predict the expected power production on a farm. The attention mechanism offers insights into the model's learning, with it seemingly capturing turbine dependencies that are aligned with physical intuition on wake losses. Similarly, Yu et al. (2020) utilize a so-called Superposition Graph Neural Network, where a temporal superposition of spatial graphs is used to predict the generated power of four different wind farms. Park and Park (2019) employ a physics-informed GNN (PGNN, introducing the physics through an engineering wake interaction model as a basis function) to estimate the power outputs of all wind turbines in any layout under any wind conditions. Bleeg (2020), on the other hand, attempts to estimate the turbine interaction loss, taking into account both wake and blockage effects using a generic GNN. They show that their proposed model combines the speed of wake models with precision that is comparable to a high-fidelity RANS flow model. Similarly, based on RANS wake data generated using a generalised actuator disk (GAD) model coupled with the CFD solver package OpenFOAM, Li et al. (2022) utilize GraphSAGE (Graph SAMPLE and aggregatE, Hamilton et al. (2017)) for flow field

reconstruction. This graph deep learning surrogate model was able to simulate wind farms within seconds, in comparison to several hours of parallel computing usually needed for a RANS simulation. Finally, Mylonas et al. (2021b) combine GNNs with Variational Bayes in a so-called Relational Variational Autoencoder (RVAE) approach to estimate farm-wide wind deficits and compare with simulated results using a steady-state wind farm wake simulator.

3.4. Approaches to transfer learning and fine-tuning

Transfer learning has become an important part of machine learning research, driven by the need for versatile frameworks which can leverage knowledge from one domain to improve performance in another (Tan et al., 2018). Transfer learning refers to methodologies which, in general, aim to leverage learned low-level features from a source domain in order improve performance in a target domain which is often much sparser. Popular methods for transfer learning in which the source and target distributions are different include Transfer Component Analysis (Pan et al., 2010) and Joint Distribution Adaptation (Long et al., 2013). When the source and target domains are very similar, or when the target domain is a more detailed subset of the source domain, one may think of transfer learning in terms of fine-tuning. With fine-tuning, the aim is to retain most of the learned behavior and only adjust certain parts of the model. This is the case for our problem, where we first aim to learn the underlying physics with lower-fidelity data, and then fine-tune on specific configurations using higher-fidelity data. This pretrain-then-finetune paradigm has recently become a popular configuration with the advent of generalist Large Language Models (LLMs) that are costly to retrain (Radford et al., 2018; Brown et al., 2020). It allows developers to leverage powerful models for niche tasks which were previously too specific to justify the costs associated with training such powerful models. Usually, fine-tuning updates all the learnable parameters of a model (Dodge et al., 2020), or sometimes certain specific layers (Guo et al., 2019). However, the increase in fine-tuning efficiency is an active field of research with several popular techniques having recently emerged (Ding et al., 2023). One such method, which we experiment with in the present work and which we detail later, is Low Rank Adaptation (LoRA) (Hu et al., 2021).

Transfer learning is especially interesting in the context of population-based SHM (PBSHM) (Worden et al., 2020; Hughes et al., 2023), allowing higher-level monitoring of structures by utilizing existing data from a source structure to improve knowledge about a target structure, for which data is sparse. GNNs in particular have shown promise for transfer learning in PBSHM, enabling researchers to efficiently extrapolate properties across populations of structures such as trusses (Tsialiamanis et al., 2022). For wind energy, several studies which apply transfer learning exist, where these often involve adapting a model trained on a specific wind farm for use on a different farm (Zraggen et al., 2021; Yin et al., 2021). Finally, with respect to PBSHM of individual turbines, Black et al. (2022) applied a neural network, trained on measured gearbox data from one turbine, to monitor the health of a different turbine within the same farm.

4. Data

4.1. Low fidelity simulations via PyWake

4.1.1. Random wind farm layout generation

To produce a robust pre-training dataset, with many different arbitrary layout geometries, we first need to construct an arbitrary wind farm layout generation methodology. We create layout geometries for wind farms using a random sampling method with four basic geometries: rectangles, triangles, ellipses, and sparse circles. To do so, we first select a minimum distance between turbines, which depends on the rotor diameter (*c.f.* Table 3). Then, a set of initial points for each random base geometry is generated, equally spaced and then randomly perturbed and rotated by a random angle, using O'Neill's permutation congruential generator (PCG, (O'Neill, 2014), as implemented by `numpy.random`). An example of the random layout geometry generation is shown in Figure 2 for a set of wind turbines, with the four

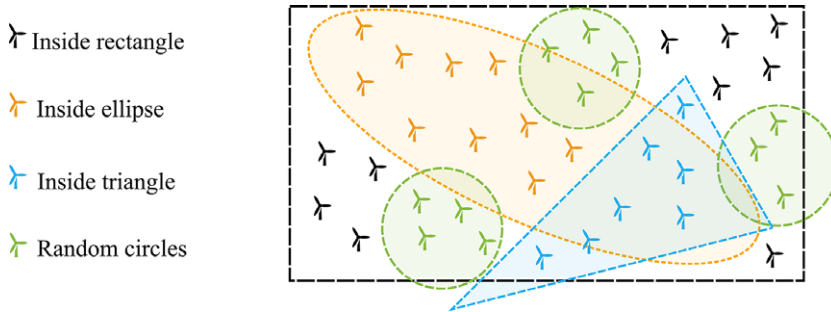


Figure 2. Illustration of random generation of wind farm layouts through pooling points inside different basic geometries (rectangle, triangle, ellipse, and sparse circles), randomly perturbed and rotated.

geometrical shapes superimposed. To produce a unique layout, only one of the four defined geometries is chosen for each training sample set, though the resulting layout is arbitrary.

4.1.2. Sampling inflow conditions

To conduct farm-wide aerodynamic simulations using PyWake, one must specify the far-field turbulent inflow wind conditions. These conditions are characterized by several random variables, namely the mean wind speed (u), turbulence intensity (T_i), wind shear (α), and horizontal inflow skewness (Ψ) (Avendaño-Valencia et al., 2021). The mean wind speed is modeled using a truncated Weibull distribution, defined by its scale and shape parameters, respectively, as:

$$\lambda_u = \frac{2 \times \mathcal{E}(u)}{\sqrt{\pi}}, \quad \text{with } \mathcal{E}(u) = 10 \quad (1)$$

$$k_u = 2.0. \quad (2)$$

The wind turbine design standard (Commission et al., 2005) describes a Normal Turbulence Model which defines the conditional dependence between the turbulence, σ_u and the mean wind speed, u . The elected reference ambient turbulence intensity (its expected value at 15 m.s^{-1}) was $I_{ref} = 0.16$. This dependency is given by the local statistical moments of the log-normal distribution (Johnson et al., 1995), given by $\sigma_u \sim \mathcal{LN}(\mu_{\sigma_u}, \sigma_{\sigma_u}^2)$, with:

$$\mathbb{E}(\sigma_u|u) = I_{ref} \cdot (0.75u + 3.8) \quad (3)$$

$$\mathbb{V}(\sigma_u|u) = (1.4I_{ref})^2 \quad (4)$$

$$\mu_{\sigma_u} = \ln \left(\frac{\mathbb{E}(\sigma_u|u)^2}{\sqrt{\mathbb{V}(\sigma_u|u)^2 + \mathbb{E}(\sigma_u|u)^2}} \right) \quad (5)$$

$$\sigma_{\sigma_u}^2 = \sqrt{\ln \left(\frac{\mathbb{V}(\sigma_u|u)}{\mathbb{E}(\sigma_u|u)} \right)^2 + 1}, \quad (6)$$

and we define the turbulence intensity as $T_i = 100 \cdot (\sigma_u)/u$. As for the wind profile above ground level, this is described by a power law relationship that is adopted to express the mean wind speed u at height Z above ground as a function of the mean wind speed at hub height u_h , measured at hub height Z_h :

$$\frac{u}{u_h} = \left(\frac{Z}{Z_h}\right)^\alpha, \tag{7}$$

where α is the shear exponent, of constant value. We sample the wind shear exponent from a truncated normal distribution (Casella and Berger, 2021), $\alpha \sim \mathcal{N}(\mu_\alpha, \sigma_\alpha^2)$ with bounds $[-0.099707, 0.499414]$.

The conditional dependence between the wind shear exponent, α , and the mean wind speed, u , is given by Dimitrov et al. (2015):

$$\mathbb{E}(\alpha|u) = 0.088 \cdot (\log(u) - 1) \tag{8}$$

$$\mathbb{V}(\alpha|u) = \left(\frac{1}{u}\right)^2. \tag{9}$$

Additionally, a custom conditional dependence between the inflow horizontal skewness, Ψ , and the mean wind speed, u , is defined by the truncated normal distribution $\Psi \sim \mathcal{N}(\mu_\Psi, \sigma_\Psi^2)$, with bounds $[-6, 6]^\circ$:

$$\mathbb{E}(\Psi|u) = \ln(u) - 3 \tag{10}$$

$$\mathbb{V}(\Psi|u) = \left(\frac{15}{u}\right)^2. \tag{11}$$

Finally, the wind direction was sampled from a uniform distribution (Dekking et al., 2005), $\Omega \sim \mathcal{U}_{[0,360]}$, spanning the full $[0, 360]^\circ$ interval. The full settings used for the inflow variables generation can be found in Table 1.

We choose to sample the turbulent inflow wind field variables (u , T_i , α , Ψ and Ω) through Sobol Quasi-Random sequences (Sobol, 1967), of $n = 2^m$ low discrepancy points in $[0, 1]^d$. These generate uniformly distributed samples over the multi-dimensional input space (unit hypercube, Saltelli et al. (2010)). In Figure 3, we can observe the matrix plot of dependence between each inflow variable's samples.

4.1.3. PyWake simulator setup

The simulations used for GNN pre-training are generated using PyWake (Pedersen et al., 2019), a multi-fidelity wind farm flow simulation tool. Utilizing Python and the `numpy` package, PyWake offers an interface to various engineering models, and is particularly efficient at analyzing wake propagation and

Table 1. Wind farm freestream inflow random variables

Parameter	Value	Description
$\mathbb{E}(u)$	10	Mean wind speed[m.s ⁻¹]
k_u	2.0	Shape factor of the Weibull distribution for wind speed
I_{ref}	0.16	Iref parameter
u_{min}	3	Minimum simulated wind speed[m.s ⁻¹]
u_{max}	25	Maximum simulated wind speed[m.s ⁻¹]
α_{min}	-0.099707	Minimum shear exponent (from PyWake)
α_{max}	0.499414	Maximum shear exponent (from PyWake)
Ψ_{min}	-6	lower limit of skewness distribution[deg]
Ψ_{max}	6	upper limit of skewness distribution[deg]
Ω_{min}	0	lower limit of wind direction distribution[deg]
Ω_{max}	360	upper limit of wind direction distribution[deg]

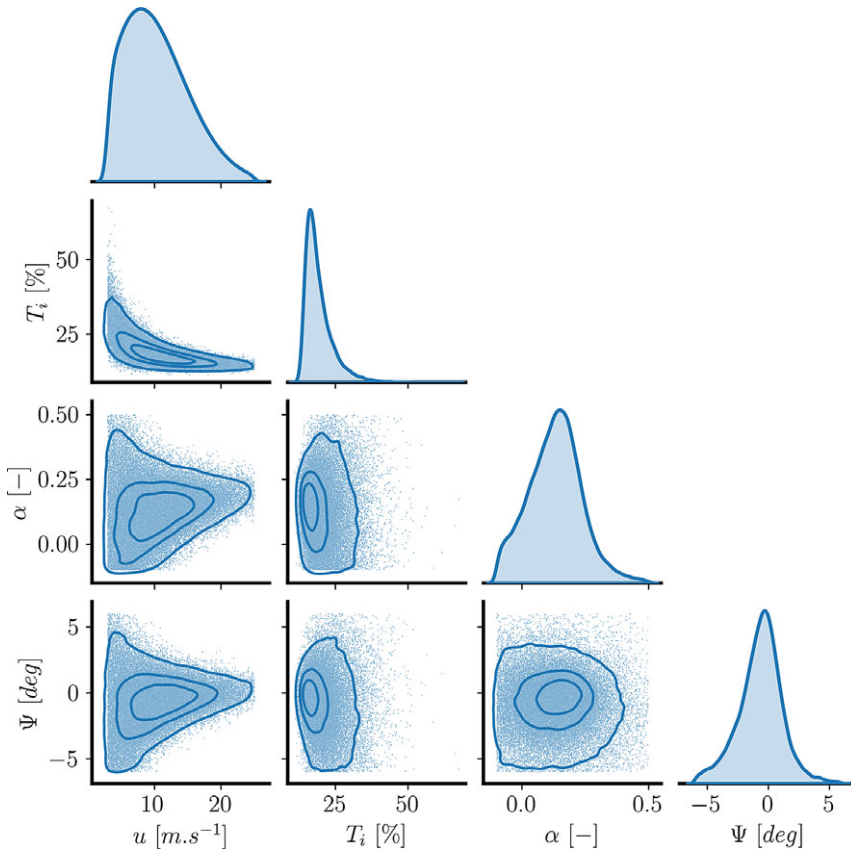


Figure 3. Samples from the joint wind inflow random variables. Non-diagonal plots additionally present a 4-level bivariate distribution through a Gaussian kernel density estimate (KDE) (Chen, 2017).

turbine interactions within a wind farm or even farm-to-farm (Fischereit et al., 2022). Its goal is to provide a cost-effective way to calculate wake interactions under various steady state conditions, and it can be used to estimate power production while accounting for wake losses in a particular wind farm layout. This makes it particularly well-suited for the quick generation of a large farm-wide dataset, albeit at reduced precision than, for instance, aeroelastic wind farm simulators. We use PyWake to produce our base dataset, with thousands of random farm layouts and inflow configurations.

Due to its modularity, PyWake allows the use of a number of engineering models. In this particular application, we adopt Niayifar's Gaussian wake deficit model (Niayifar and Porté-Agel, 2016). Niayifar proposes a wake expansion/growth rate k that varies linearly with local turbulence intensity:

$$k = a_1 I + a_2, \quad (12)$$

with I , the local streamwise turbulence intensity immediately upwind of the rotor center, estimated neglecting blockage effects, and the coefficients $a_1 = 0.3837$ and $a_2 = 0.0036$ based on Large-Eddy Simulations (LES) of Bastankhah and Porté-Agel (2014).

The interaction among multiple wakes is modeled based on the velocity deficit superposition principle, with the velocity deficit calculated as the difference between the inflow velocity at the turbine and the wake velocity:

$$U_i = U_\infty - \sum_{k=1}^n (U_k - U_{ki}). \quad (13)$$

Table 2. PyWake outputed quantities

Sensor	Units	Description
u_{loc}	$m.s^{-1}$	Rotor averaged wind speed
Ti_{loc}	%	Turbulence intensity
P_{loc}	kW	Mean power output
DEL_{flap}	Nm	Blade root flapwise damage equivalent load
DEL_{edge}	Nm	Blade root edgewise damage equivalent load
DEL_{tor}	Nm	Tower top torsional damage equivalent load
DEL_{fa}	Nm	Tower base fore-aft damage equivalent load
DEL_{ss}	Nm	Tower base side-side damage equivalent load

Regarding the turbulence intensity model, we employ the Crespo-Hernández’s model (Crespo and Hernández, 1996), where the added turbulence intensity is given by:

$$\Delta I = 1.026 \frac{\Delta k^{0.5}}{U_{\infty}}. \tag{14}$$

As for the fatigue load estimation, PyWake implements in its two wind-turbine approach a neural network surrogate of HAWC2 simulations (Larsen and Hansen, 2007) in different wake and inflow conditions, where the wind farm is split into pairs of wind turbines and returns the load of the pair that yields the highest load. PyWake can report for each turbine in a simulated farm 8 ‘sensor’ quantities which we describe in Table 2. In our analysis, we use the International Energy Agency’s (IEA) 3.4MW reference wind turbine (RWT, Bortolotti et al. (2019)), as described in Table 3.

Utilizing the described pipeline, we synthesize a dataset of 3,000 randomly selected wind farm layouts, with each layout being subjected to a unique set of 10 different random inflow conditions. Consequently, this process yields a total of 30,000 unique graphs that can be used to pre-train our GNN model. The full dataset can be found in de N Santos et al. (2023a).

4.2. High fidelity aeroelastic simulations

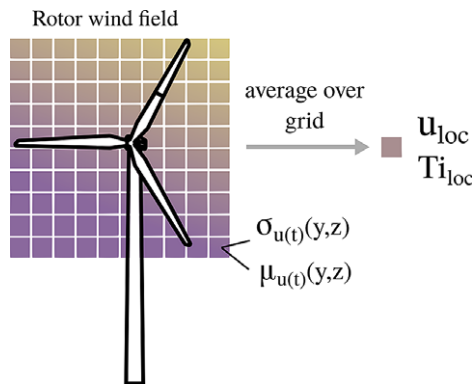
To develop and test our multi-fidelity transfer framework, we make use of a dataset of aeroelastic simulations generated by Liew et al. (2023a). The authors present a comprehensive database of 1000 one-hour long simulations of the Lillgrund offshore wind farm computed using HAWC2Farm. This farm is composed of 48 2.3MW Siemens SWT-2.3-93 turbines, the characteristics of which can be found in Table 4. HAWC2Farm models each individual turbine of a wind farm using the HAWC2 aeroelastic simulator (Larsen and Hansen, 2007), with their interactions being computed using a dynamic wake meandering model (Larsen et al., 2008). Here, the simulations are performed at a sampling rate of

Table 3. Characteristics of the IEA 3.4MW reference wind turbine used in PyWake

Item	Value	Units
Rated Power	3370	kW
Rated Wind Speed	9.8	$m.s^{-1}$
Cut-in Wind Speed	4	$m.s^{-1}$
Cut-out Wind Speed	25	$m.s^{-1}$
Rotor Diameter	130	m
Hub Height	110	m

Table 4. Characteristics of the Siemens SWT-2.3-93 wind turbines of the Lillgrund wind farm

Item	Value	Units
Rated power	2,300	kW
Rated wind speed	13	m.s^{-1}
Cut-in wind speed	3.5	m.s^{-1}
Cut-out wind speed	25	m.s^{-1}
Rotor diameter	93	m
Hub height	65	m

**Figure 4.** Illustration of inference of local flow quantities from the vertical grid in front of each turbine for the HAWC2Farm dataset.

100 Hz using a large turbulence box which covers the entire farm. In the box, turbulence is generated using the presented Mann turbulence model implementation. The authors use the Halton sequence (Halton, 1960) to sample a range of ambient wind speeds from 4 m/s to 20 m/s, turbulence intensities from 2.5% to 10%, wind directions from 0 to 360 degrees, and shear exponents ranging from 0 to 0.3. The dataset, thus, captures the dynamic structural response and fatigue loading of the 48 turbines in the farm across a reasonably wide range of operating conditions.

For our testing, we use a subset of 800 of these simulations which was made publicly available (Liew et al., 2023b). In this dataset, the local flow conditions are parameterized by a 30x30 grid in front of each turbine rotor, where each cell of this grid contains the average and the standard deviation of the flow gathered over 10 minutes. To obtain comparable local flow outputs to PyWake, we therefore first have to convert the velocity grids. Taking the mean of the time-averaged velocity over this grid gives an estimation of the local value of the wind speed, while the local turbulence intensity can be computed by dividing the mean velocity by the mean over the grid of the time-gathered standard deviation of each cell. Figure 4 illustrates this process. The remaining variables that we require can be directly extracted from the 10-minute average HAWC2 outputs included in the dataset. The initial HAWC2 simulations have a duration of 1 hour each, yielding six 10 minute periods of averaged quantities for each variable and each simulation. We only keep the first such period of each simulation, as the difference in the resulting averaged variables between these periods is minimal.

4.2.1. Visualizing wake effects and model disparities

In Figure 5 we plot, for the Lillgrund wind farm layout and a predominant wind direction N-NE (Ω), the HAWC2Farm and PyWake outputs for two turbines under different flow conditions: WT 31 (blue, free-flow), WT 35 (orange, wake-affected). In this figure, we observe how the wake-affected turbine, which is located in

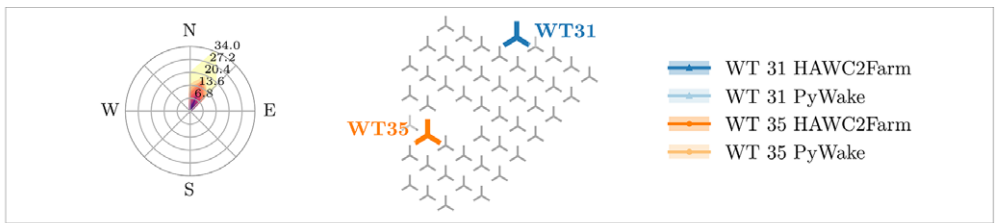


Figure 5. PyWake and HAWC2Farm outputted quantities vs. global far-field wind speed for two wind turbines of the Lillgrund layout under contrasting wake conditions, for a range of N-E inflows. The simulated turbines are different for the two simulators, yielding vastly different loads. Shaded curves represent the 75% and 95% distribution percentiles.

the middle of the farm under strong wake conditions, present much higher damage-equivalent loads, more evident for fore-aft, torsional and flapwise DELs, which makes physical sense. There are large differences in the magnitude of the DELs, with HAWC2 loads being dwarfed by those from the PyWake simulations. This occurs due to the different turbine models used in each simulator. Despite these differences, the underlying physics are similar, as highlighted by the behavior of the local turbulence intensity, wind speed and power.

Aside quantitative differences, there are, however, qualitative differences which exemplify the lower ability of PyWake, namely the offsets present for the blade fatigue loads (edge- and flapwise), along with serious jumps, as seen for fore-aft and side-to-side. The observed differences stem from the different levels of simulation fidelity between PyWake and HAWC2. While PyWake employs engineering wake-expansion models coupled with a machine learning-based surrogate for aeroelastic load estimation, HAWC2 is a comprehensive aeroelastic code that conducts time-domain simulations to capture a wind turbine's dynamic response. Thus, the loads computed via HAWC2Farm are inherently more nuanced than those given by PyWake.

5. Graph neural network framework

5.1. Establishing the graphs

A wind farm can be modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E}, \mathbf{W})$, where $\mathcal{V}$ represents a set of n nodes (individual turbines), and $\mathcal{E}$ is the set of k edges which represent the interactions between turbines. Each turbine node i is associated with a feature vector $\mathbf{x}_i \in \mathbf{X}$, containing f_n physical measurements and/or properties (e.g., power, loads, local effective turbulence intensity). For each edge connecting turbines i and j , an attribute vector $\mathbf{e}_{i,j} \in \mathbf{E}$ is defined, encapsulating the f_e geometrical properties of the edge. The wind farm graph also includes a global property vector $\mathbf{W} \in \mathbb{R}^{f_g}$, of size 3, to represent farm-level inflow conditions. $\mathbf{X} \in \mathbb{R}^{n \times f_n}$, and $\mathbf{E} \in \mathbb{R}^{k \times f_e}$ are respectively the node and edge feature matrices of this graph representation. This section details how the edge connectivity is constructed and describes the input and output attributes of the graphs.

5.1.1. Graph connectivity

Various methods could be used to determine the graph edge connectivity $\mathcal{E}$ for a specific wind farm layout. In this work, we choose to employ a widely-used meshing technique for the generation of unstructured triangular meshes, the Delaunay triangulation (Delaunay et al., 1934; Musin, 1997). Unlike other popular point-to-graph generation techniques (radial, k-nearest-neighbours), this method ensures that no isolated communities are formed, i.e. that the generated graph is *connected*. Moreover, previous results (Duthé et al., 2023b) have shown that against the fully-connected method, in which all vertices of a graph are connected to each-other, Delaunay triangulation yields comparable results at a much lesser computational cost. We show in Figure 6 some examples of wind farm graphs connected using this method.

5.1.2. Graph features

One of the two main inputs to our model is the ensemble of variables which characterize the inflow conditions. We treat these inflow variables, namely the wind direction Ω , the wind velocity u , and the

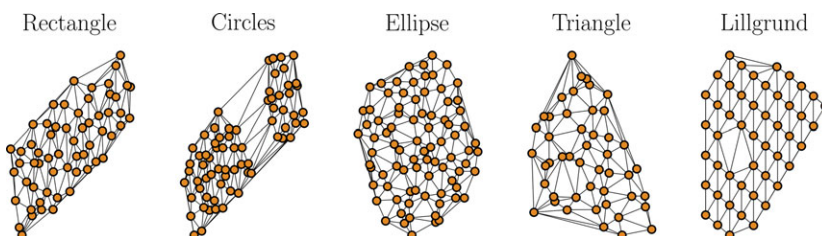


Figure 6. Delaunay triangulation of 4 randomly generated layouts and of the real Lillgrund offshore wind farm.

turbulence intensity T_i , as a vector of global attributes $\mathbf{W}$, i.e. attributes that have an effect on all nodes of the graph $\mathcal{G}$. The other main input to our model is the geometric description of a given wind farm. This information is passed to the model via a vector of attributes $\mathbf{e}_{i,j}$ for each edge $e_{i,j} \in \mathcal{E}$ which describes the relative positions of the nodes. In particular, each edge connecting two nodes i and j has an attribute vector that stores three quantities: the distance between two nodes $d_{i,j}$, the angle of the edge with regard to the global north $\alpha_{i,j}$ and the angle of the edge with regard to the inflow direction $\beta_{i,j}$.

We treat the problem as a node regression task, where the model outputs 8 target features for each node. For each node $v_i \in \mathcal{V}$, the features of the output vector $\mathbf{x}_i$ following: power P_i , local wind speed $u_{loc,i}$, local turbulence intensity $Ti_{loc,i}$, blade flapwise DEL $MyBr_i$, blade edgewise DEL $MxBr_i$, fore-aft tower DEL $MyTB_i$, side-to-side tower DEL $MxTB_i$ and tower top torsional DEL $MzTT_i$. The input and output graphs and their features are displayed in Figure 7.

All features—both input and target—are normalized before training. We use the statistics (mean and standard deviation) computed on the entire training set to standardize each variable.

5.2. GNN architecture

The objective of this framework can be treated as a multivariate node regression task, where the graph connectivity and graph-level attributes are given, but where no node information is known a priori:

$$X_{estimated} = \text{GNN}(\mathcal{V}, \mathcal{E}, \mathbf{E}, \mathbf{W}) \quad (15)$$

In this section, we describe the individual components of the framework required to perform this task.

5.2.1. Encode-Process-Decode structure

Building on the approach outlined in Sanchez-Gonzalez et al. (2020), our study also employs an Encode-Process-Decode GNN architecture. This architecture choice facilitates message-passing operations in a latent space characterized by significantly higher dimensionality compared to the physical parameter space. Such increased dimensionality notably enhances the GNN's capacity for complex representation, therefore increasing its potential expressivity. This design allows the network to capture intricate patterns and relationships in the data, which could be particularly advantageous for complex physics-based problem-solving tasks, such as predicting non-linear wake-induced loads. A visual representation of the proposed GNN method is shown in Figure 8.

5.2.2. Encoding into a high-dimensional latent space

To operate within an N -dimensional latent space, graph attributes initially undergo a preprocessing step via an encoding layer. This layer comprises three distinct ReLU-activated Multilayer Perceptrons (MLPs),

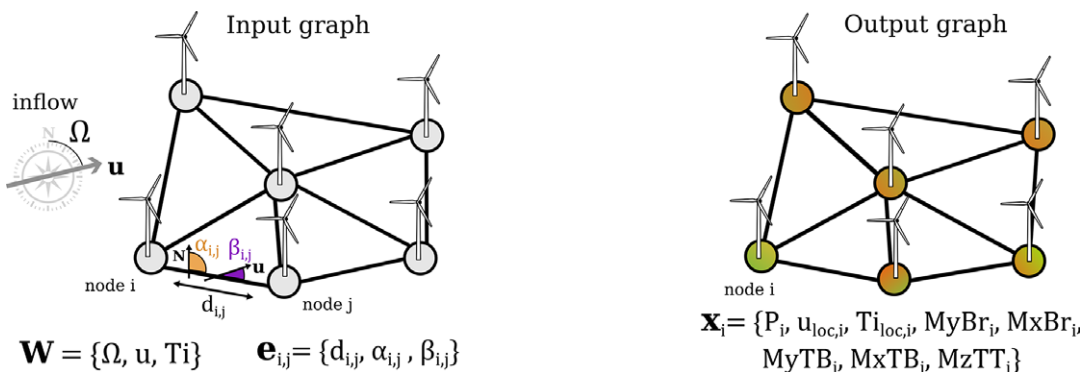


Figure 7. Features of the input and output graphs. At the input, there are no node features only global inflow features and geometrical edge features. At the output, we obtain 8 features for each turbine node.

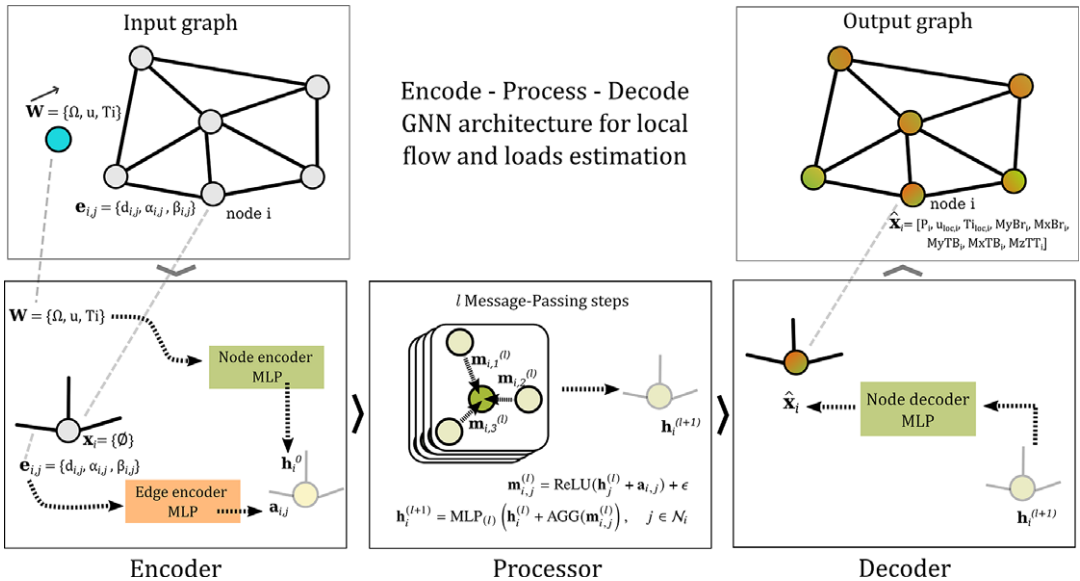


Figure 8. Graphical overview of the proposed GNN framework. We use an Encode-Process-Decode structure with GEN message-passing. Each message-passing layer within the Processor uses a separate MLP to update the latent node features.

each containing two hidden layers. Specifically, the Edge-Encoder processes the edge features, transforming them into encoded edge vectors denoted as $\mathbf{a}_{ij}$. Concurrently, the Node-Encoder focuses on the inflow global vector $\mathbf{W}$ to produce the latent node features $\mathbf{h}_i$. The encoding process can be therefore be expressed as:

$$\mathbf{h}_i^{(0)} = \text{MLP}_{\text{ENC,node}}(\mathbf{W}), \quad \mathbf{a}_{ij} = \text{MLP}_{\text{ENC,edge}}(\mathbf{e}_{ij}) \quad (16)$$

5.2.3. Message-passing processor

The message-passing procedure serves as the foundational element in the GNN algorithm (Gilmer et al., 2017). This involves a dual-phase approach for updating node attributes. Each message-passing layer of the Processor first calculates messages $\mathbf{m}_{ij}$ across all edges; these are then aggregated at nodes using pooling functions such as max or mean. Algorithm 1 provides an overview of these steps.

Algorithm 1 Message-passing layer l

for $e_{ij} \in \mathcal{E}$ do	$\triangleright$ For each edge, compute a message
$\mathbf{m}_{ij}^{(l)} \leftarrow \phi(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}, \mathbf{a}_{ij})$	$\triangleright \phi$ is the message update function, often a neural network
end for	
for $v_i \in V$ do	$\triangleright$ For each node, update the features
$\mathbf{agg}_i \leftarrow \text{AGG}(\mathbf{m}_{ij}^{(l)}, j \in \mathcal{N}_i)$	$\triangleright$ Permutation-invariant aggregation of messages (e.g. sum)
$\mathbf{h}_i^{(l+1)} \leftarrow \varphi(\mathbf{h}_i^{(l)}, \mathbf{agg}_i)$	$\triangleright \varphi$ is the node update function, often a neural network
end for	

In this work, we adopt the GENeralized Aggregation Networks (GEN) (Li et al., 2020a) message-passing setup. GEN is a variant of the popular Graph Convolution Network (Welling and Kipf, 2016),

which allows for edge features and other more advanced message aggregation AGG functions. The GEN message-passing update for layer l writes:

$$\mathbf{m}_{ij}^{(l)} = \text{ReLU}\left(\mathbf{h}_j^{(l)} + \mathbf{a}_{ij}\right) + \varepsilon, \quad (17)$$

$$\mathbf{h}_i^{(l+1)} = \text{MLP}_{(l)}\left(\mathbf{h}_i^{(l)} + \text{AGG}\left(\mathbf{m}_{ij}^{(l)}\right)\right), \quad j \in \mathcal{N}_i. \quad (18)$$

where ε is a small positive constant, typically set to 10^{-7} . In our implementation we use the `softmax` type aggregation function, which can be expressed as:

$$\text{AGG}\left(\mathbf{m}_{ij}^{(l)}\right) = \sum_{j \in \mathcal{N}_i} \frac{\exp\left(\beta \cdot \mathbf{m}_{ij}^{(l)}\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\beta \cdot \mathbf{m}_{ik}^{(l)}\right)} \cdot \mathbf{m}_{ij}^{(l)} \quad (19)$$

We choose to employ this specific message-passing formulation, based on previous results (de N Santos et al., 2024a). When compared against two other popular architectures, the Graph Attention Network (GAT, Veličković et al. (2017)) and the Graph Isomorphism Network with Edges (GINE, Xu et al. (2018); Hu et al. (2019)), the GEN layer was found to be the best performing layer out of the three. Indeed for this specific problem, the GAT layer suffers from latent space collapse, while the GINE model appears to be too sensitive to inflow variability.

5.2.4. Projecting back onto the physical domain

After the L message-passing steps, the decoding layer functions to extract final physical quantities from the latent node features. Structurally akin to the encoder, this layer consists of a ReLU-activated MLP with two hidden layers. In essence, this step translates the high-dimensional latent features back into a physically interpretable space. The decoding step is formalized as:

$$\hat{\mathbf{x}}_i = \text{MLP}_{DEC,node}\left(\mathbf{h}_i^{(L)}\right) \quad (20)$$

5.2.5. Pre-training the baseline general GNN

Our GNN framework is implemented using the `PyTorch` (Paszke et al., 2019) and `PyTorch-Geometric` (Fey and Lenssen, 2019) python packages. Our baseline model is constructed with a latent dimensionality of 250, 4 message-passing GEN layers and 2 hidden-layer MLPs for the node and edge encoders as well as for the Node decoder. This yields a baseline model with a total of around 1,385,500 trainable parameters. The model is trained to minimize an $L2$ loss for all output variables over 150 epochs using the Adam optimizer (Kingma and Ba, 2014). The initial learning rate is set to $1e-3$ and is annealed during training using a cosine scheme (Loshchilov and Hutter, 2016).

5.3. Fine-tuning GNNs

Given a model, constructed as described in subsection 7.1, and trained on the PyWake dataset, we aim now to fine-tune this model for the dataset of Lillgrund farm simulations described in subsection 4.2. This is a challenging task, where the difficult is twofold. Not only is the HAWC2 data of a higher fidelity and complexity, but the turbines themselves are different. This difference in the model of the simulated turbine has a major impact on the absolute values of the quantities we are interested in, proper normalization is essential. Given this challenge, we chose to test three different approaches to perform the fine-tuning, comparing each against a baseline non-pretrained ‘Scratch’ model. Table 5 summarizes the properties of the different methods, which we then describe below.

5.3.1. Vanilla

The most straightforward method for fine-tuning, which we denominate as the ‘Vanilla’ method, is to simply take the pre-trained model and re-train it on the new data. In this case, no weights of the model are frozen and all layers are trained using the higher-fidelity dataset.

Table 5. Details of the different fine-tuning methods

	Scratch	Vanilla	Decoder	LoRA
Uses a pre-trained model	No	Yes	Yes	Yes
Frozen layers	None	None	Encoder, processor	All
Number of trainable Parameters	~1,385,500	~1,385,500	~127,500	~15,000

5.3.2. Decoder only

Another possibility is to fine-tune only certain predefined parts of the model. In our fine-tuning task the inputs (global wind direction, wind speed and turbulence intensity) do not change, however because of the difference in turbine models between the two datasets, the output quantities are vastly different. Therefore, it makes sense to only tune the final output network, i.e. the decoder, whilst keeping the rest of the model frozen. This method can also be thought of in terms of *feature extraction*.

5.3.3. Low rank adaptation

Low rank adaptation (LoRA) (Hu et al., 2021) is a recent method for more efficient fine-tuning, originally developed for large language models (LLMs) with billions of parameters. LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each linear layer of the model. The authors suggest that changes in the model weights during fine-tuning have a “low intrinsic rank,” and thus one can reduce the memory footprint of the model by only updating low-rank approximations of the changes Δ for all dense layer weight matrices instead of updating the full-rank matrices. Thus, for a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times p}$, its update is constrained by representing it using a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA \quad (21)$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times p}$, and r is the rank of the decomposition. For all testing we use a rank $r = 4$. Throughout the fine-tuning process, W_0 is frozen and does not receive gradient updates, whilst A and B both contain trainable parameters. During the forward pass, both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise. For $h = W_0 x$, the modified forward pass in LoRA would be:

$$h = W_0 x + (BA)x \quad (22)$$

This mathematical formulation allows LoRA to be memory-efficient while maintaining or even improving performance compared to full fine-tuning (Vanilla). As LoRA is a linear parametrization, for deployment the trainable matrices can be merged with the frozen pre-trained weights, thus yielding no additional inference overhead.

5.3.4. Fine-tuning implementation

Again, we use PyTorch to implement the different fine-tuning methods. All fine-tuned models are trained by minimizing a $L2$ reconstruction loss between the HAWC2Farm outputs and the GNN outputs. We train the models for 200 epochs using an initial learning rate of $5e-3$ that is annealed during training with a cosine scheduler. For the LoRA model, we employ a minimal implementation of the original method, named minLoRA (Chang, 2023).

6. Results

In the present section, we proceed to analyse the expressiveness of GNNs in capturing fundamental physical behaviour on wind farms. We initiate by assessing the base performance of the pre-trained GNN

model on the lower resolution PyWake data. We then focus on the fine-tuning performance, specifically addressing which approach (LoRA, Vanilla, Decoder or Scratch) is preferable, as a function of the amount of available training data. We then discuss in detail the results in relation to the each predicted quantity and the overall farm fatigue patterns. Finally, we identify relative pitfalls, and study possible mitigation strategies.

6.1. GNN predictions for PyWake data

We first aim to confirm that the baseline model, trained on the PyWake dataset, sufficiently captures the simulated behavior of a wind farm. In Figure 9, we plot for an unseen random wind farm layout, the different outputted quantities of the GNN against those given by the PyWake simulator for two turbines in a range of inflow conditions in all wind directions (every 5° from 0° to 360°). We observe that the GNN predictions closely align with those given by PyWake.

This polar plot is of particular interest as it not only demonstrates a good agreement between the GNN model predictions and the simulated PyWake quantities, but further allows deeper insights into the farm wake physics and how the GNN model is capturing these. This is exemplified by the preponderance of N–SE sector (0°–140°) for WT51 in its turbulence intensity. It is in this region where WT51 sees its highest turbulence intensity. This has, as expected, an impact on the fatigue loads which, for pure aerodynamic simulations (*e.g.*, no hydrodynamics) are strongly correlated with wake and, therefore,

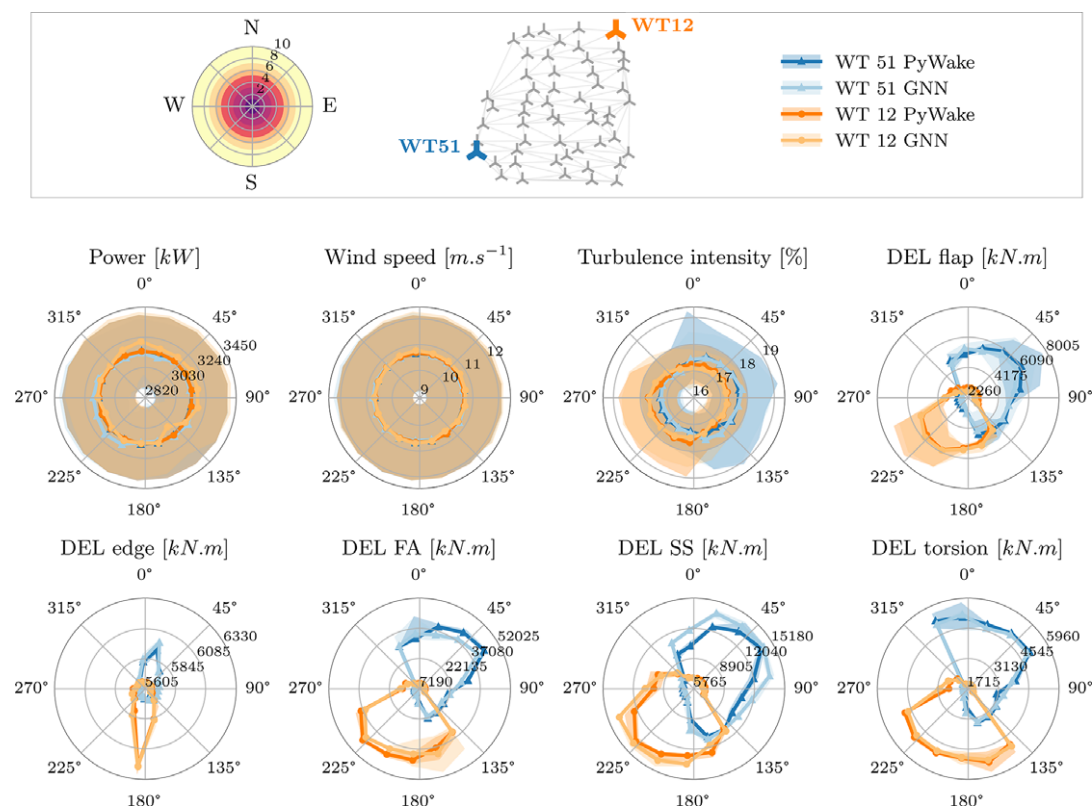


Figure 9. Polar plots of the outputs of the baseline GNN (trained only on the PyWake data) versus the PyWake simulation outputs as a function of wind direction for an unseen random farm layout with 720 inflow conditions (10 inflow velocities per 5° sector). We show the outputted quantities for two turbines on opposite sides of the farm (WT 51 in blue and WT 12 in orange). Shaded surfaces indicate 5% variance around the mean.

turbulence intensity. It ought however be noticed that fatigue loads are strongly non-linear, as evidenced by the plot, and phenomena like accumulating nature of the wake effect. Other loads, like the blade-root edgewise moment, which is mostly gravitational-dependent but is also influenced by some aerodynamic loads (Rinker et al., 2021), depend more on directly near-wake turbines than far-wake accumulation over the farm and are, therefore, less related to overall turbulence intensity. As for WT12, as it is diametrically opposite to WT51, its pattern of wind direction dependency for turbulence intensity and fatigue loads almost perfectly mirrors WT51 (180° inversion).

6.2. Fine-tuning

Here, we discuss in detail the different approaches for fine-tuning and their expressiveness in relation to transfer learning. As a quantitative metric for most of our results we use the Mean Absolute Percentage Error (MAPE), which can be formulated as a function of the actual values, represented as y , and the predicted values, represented as $\hat{y}$. The MAPE function is defined as follows:

$$\text{MAPE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100\% \quad (23)$$

where y_i denotes the actual value and $\hat{y}_i$ denotes the predicted value for the i -th observation out of N total observations. One must however be aware that when the true value lies close to zero, even small errors may lead to disproportionately large percentage errors. In our case, the only predicted value which may be zero is power, as such when computing the MAPE for power we exclude cases where no power is produced.

6.2.1. Overall performance of the different fine-tuning methods

Given the four different approaches described in Table 5, we can compare their relative performance. Here, it is relevant to highlight that the Scratch approach, wherein there is no transfer learning and no use of a pre-trained model (the GNN is trained directly on the available data), serves as the baseline to which each model compares itself. The publicly available dataset described in subsection 4.2 is comprised of 800 HAWC2 aeroelastic simulations. Of these, 200 simulations are saved to test the performance of the trained models. The rest of the available data is pooled and used to train the model in different sample sizes (2–500), to demonstrate the effectiveness of each method given a certain number of training samples and to discern what is the minimal amount of samples required to reach a set prediction error.

In Figure 10, we perform a comparison of each method for different training sample sizes by analyzing their MAPE, aggregated for all 8 outputted variables. For every sample size, all methods are subjected to ten evaluations, with each evaluation representing a separate training run. These runs are distinct due to individual seeds that dictate the random selection of the training samples.

In this figure, we observe that all models converge to the same error with increasing sample sizes (around 10%). We can further observe how all methods (LoRA, vanilla and decoder) outperform training a model from scratch at all sample sizes bar the last (where the decoder models perform slightly worse). There is interestingly a inversion of the different methods' performance around 100 samples (highlighted region, in violet): for a low amount of training samples, LoRA and the decoder-only retraining methods perform the best. However, with greater amounts of data the Decoder model stagnates, whereas the Vanilla and LoRA models continue improving. This makes sense as the number of trainable parameters of the Vanilla model are ten times greater than for decoder-only. With only the readout layer unfrozen, the decoder-only model seemingly hits a limit in its representation power for the HAWC2 dataset. Similarly, LoRA models are highly performant, as low-rank adaptation has been developed to effectively deal with billions of parameters of large models. We identify this inflection point in Figure 10 by highlighting in purple the models for 106 training samples. This is a significant number of samples because it represents a point where the Decoder's performance stagnates and there is already a level of convergence. Additionally, this number of samples represent about half of the test dataset. In Table 6, we compare the

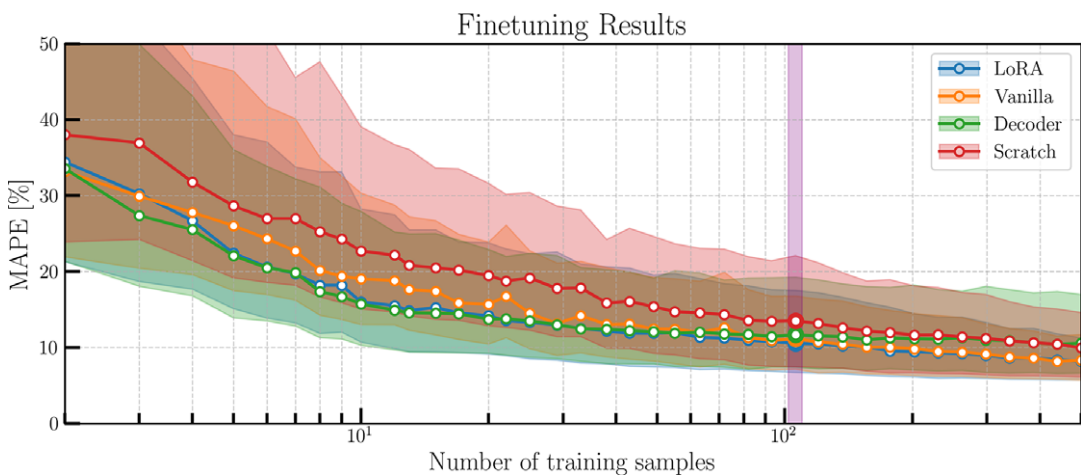


Figure 10. Comparing the different fine-tuning methods. MAPE metrics are shown as an average of each output variable, gathered on the test set with 200 samples. Solid lines indicate the mean MAPE results gathered over the 10 data splitting seeds. Violet highlight represents the 106 training samples. Shaded curves represent the 90% distribution percentile.

Table 6. Performance of the different fine-tuning methods (MAPE) for each estimated variable for two training samples sizes and averaged over all runs with different data-splitting seeds. Best performing method is highlighted for each sample size (black, 106; grey, 500). For power, MAPE scores are only computed in cases when power is produced

Variable	# Samples	LoRA	Vanilla	Decoder	Scratch
Power	106	5.28	5.25	7.71	8.65
	500	3.11	2.77	5.91	4.33
Wind speed	106	3.89	4.62	4.75	5.62
	500	2.63	2.88	4.14	3.48
Turbulence intensity	106	9.45	10.18	12.16	15.22
	500	6.54	6.66	10.50	8.57
Flapwise DEL	106	2.12	1.98	2.25	2.21
	500	1.48	1.48	2.05	1.80
Edgewise DEL	106	16.07	15.94	15.81	16.32
	500	13.73	13.61	15.37	14.95
FA DEL	106	12.05	11.62	11.96	12.55
	500	9.81	9.68	11.16	10.44
SS DEL	106	7.92	8.33	8.83	10.82
	500	5.86	6.08	7.79	7.09
Torsional DEL	106	7.49	8.19	7.80	8.00
	500	5.58	5.98	7.05	6.27

performance of each method for each estimated variable, highlighting the best-performing model at 106 (black) and 500 (grey) samples.

This table reveals both LoRA and the vanilla scheme as the best performing models with, generally, LoRA performing better for wind speed, turbulence intensity, side-to-side and torsional DELs, and the vanilla model performing better for power, flapwise, edgewise and fore-aft DELs. However, for both

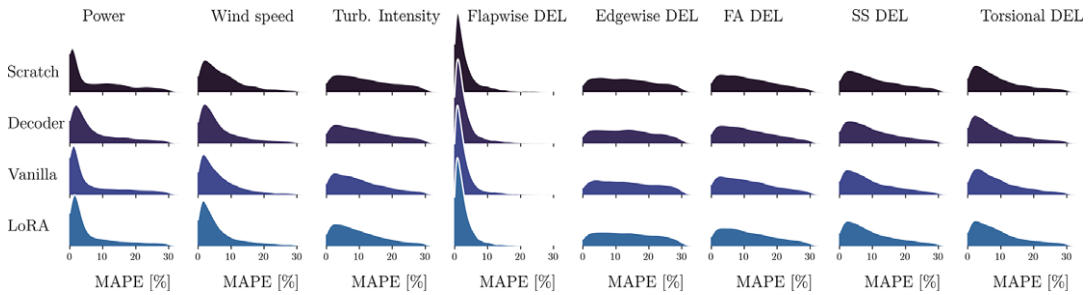


Figure 11. Comparing the different fine-tuning methods with all models trained on 106 HAWC2Farm simulations. MAPE distributions gathered on the test set with 200 samples are shown for each output variable.

sample sizes, the errors are comparable. We further observe, for both models, a significant improvement with respect to training from scratch (for 106 this difference can be +5%). Therefore, in our further analyses we focus on the best performing LoRA model for 106 samples. By analyzing a model for such a low amount of data we remain faithful to the objective of developing a generalist model which does not require great quantities of data to effectively adapt to new turbines/fidelity.

In Figure 11 we further examine the distributions of the prediction error per output quantity for all models. We can directly identify the relative difference between each predicted quantity. In general, the closer the error curves are skewed towards 0% (*i.e.*, Weibull), the easier the quantity is to predict, *e.g.*, power or flapwise DEL. Conversely, if the error resembles a continuous distribution, then the model has a greater difficulty to capture that quantity (*e.g.*, edgewise DEL). Additionally, differences between each fine-tuning method are noticeable. In general, LoRA presents error distributions more skewed towards 0% than its counterparts. For example, when predicting turbulence intensity using Scratch or the decoder the error resembles a continuous distribution, whereas LoRA is closer to a Weibull.

6.2.2. Model physicality

Comparing farm-level behavior. An integral part of validating the results lies in confirming their physicality, that is to say, if the results make physical sense. We posit that our GNN model is capable of learning underlying physical relationships (such as wakes) in a coarse level (PyWake) before being fine-tuned for a specific task on a higher-fidelity solver. Given our pre-training scheme, the GNN should first learn *relatively* before learning *specifically*. To verify whether our training regime does indeed allow the GNN to learn what a wake is, we plot in Figure 12 the predicted local effective turbulence intensity for each turbine of the Lillgrund farm for all models at 106 samples, for a given inflow condition.

In this figure, we can observe that the Scratch model predicts an average turbulence intensity value for all turbines, indicating that it is unable to identify front-row turbines, distinguishing these as free-flow and effectively learn wake effects. However, this is not the case for the pre-trained models (Decoder, Vanilla and LoRA GNNs), indicating that pre-training with PyWake does result in better underlying physicality. However, we note that the HAWC2Farm ground truth is noticeably more nuanced than the equivalent PyWake solution, especially for the back-row turbines where the effects of multiple wakes start to stack. This is something which all GNN models do not completely capture, with only the Vanilla and LoRA models approaching the HAWC2Farm solution for the non-waked turbines. LoRA is particularly similar to HAWC2 in the first two strings, but more complex phenomena like the ‘gap’ for the 3rd and 4th rows, which in HAWC2 has an impact on all subsequent rows, making the northeasterly sector of the farm more nuanced (turbulence intensities between 14% and 17%, in orange–violet), is much more binary for LoRA (just two turbines immediately after the gap appear with a lower turbulence intensity, as in PyWake). This seems to suggest that pre-training allows model to quickly understand basic physical behaviour, like the distinction between free-flow and wake-affected turbines, but more intricate wake behaviour such as the

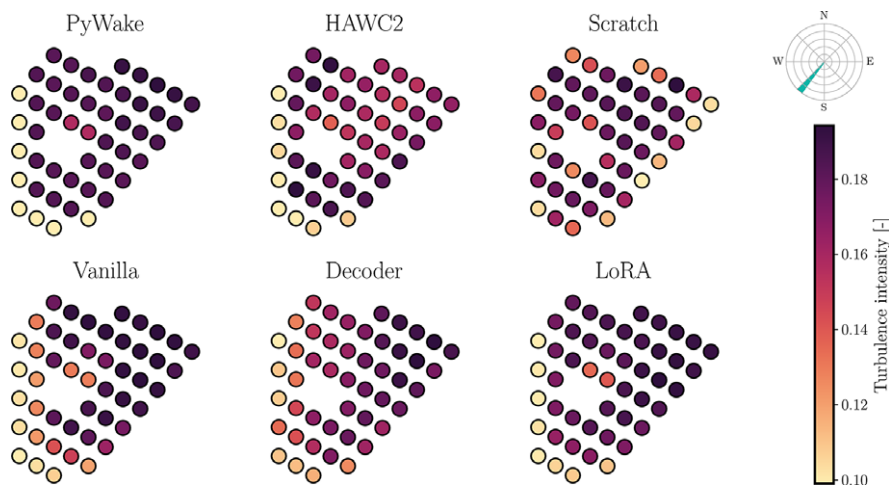


Figure 12. Comparison of the predicted local effective turbulence intensity for the different models trained on 106 HAWC2Farm simulations for a single southwesterly inflow condition (9.4 m.s^{-1} and 9.9% turbulence intensity). For reference, we also show the equivalent PyWake output.

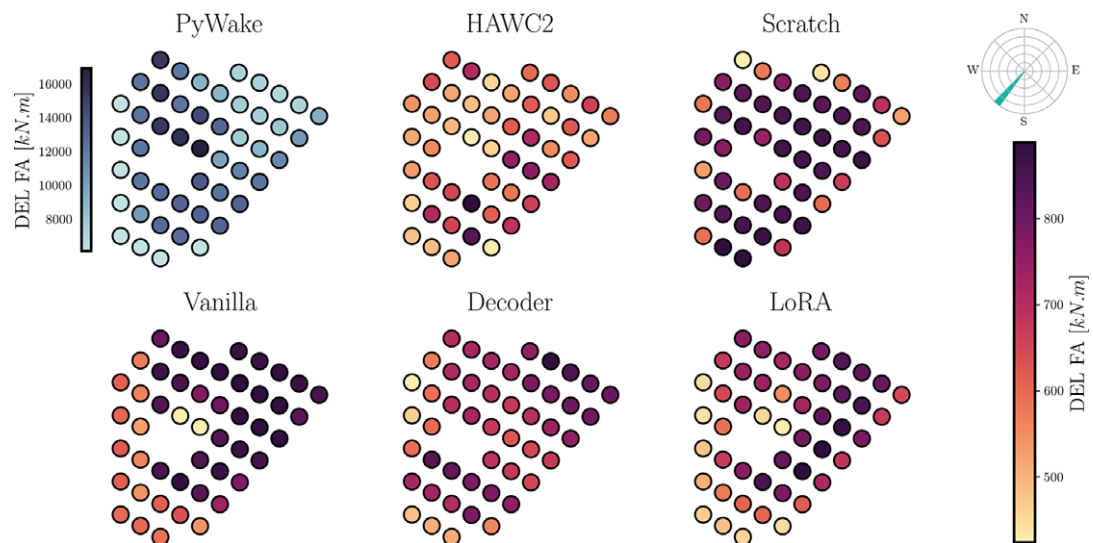


Figure 13. Comparison of the predicted fore-aft damage equivalent loads for the different models trained on 106 HAWC2Farm simulations for a single southwesterly inflow condition (9.4 m.s^{-1} and 9.9% turbulence intensity). For reference we also show the equivalent PyWake output using a different colormap due to the turbine model disparity.

interplay between far-wake and near-wake boundary layer recovery are not captured, mostly because the baseline simulator (PyWake) does not have the necessary resolution to address these situations.

This absence of nuance is also reflected in the predicted DELs, as shown in Figure 13. Predicting the DELs correctly is more challenging (as seen above, they are more non-linear), and here the PyWake legacy may actually be harmful. This may be observed by examining the discrepancies in the patterns between the HAWC2Farm and PyWake solutions, where the turbines with maximal predicted FA DELs do not match. While the fine-tuned GNN models do also lack nuance here, the extrema in the predictions

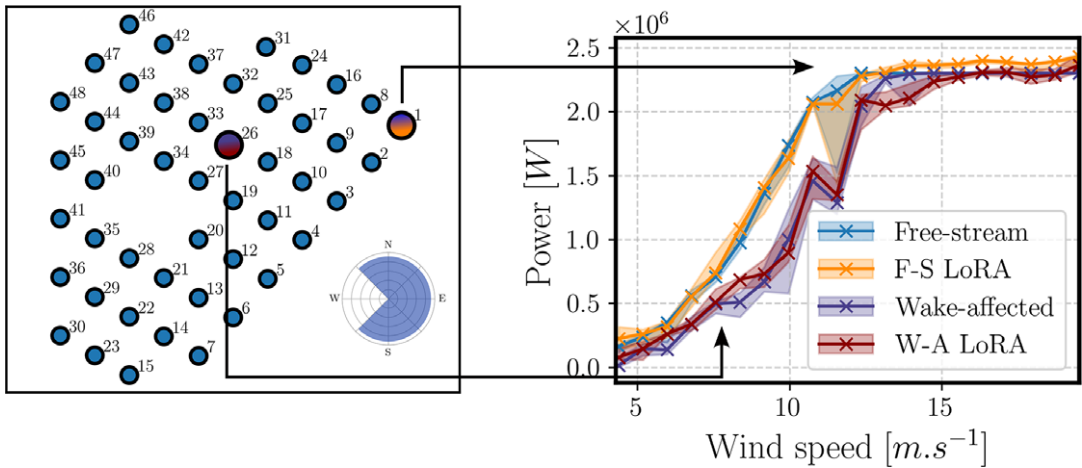


Figure 14. Power curve for free-stream turbine (WT1; blue, true; orange, predicted using LoRA) and wake-affected turbine (WT26; purple, true; dark red, predicted using LoRA). SW–NW wind direction excluded. Shaded curves represent the 75% distribution percentiles.

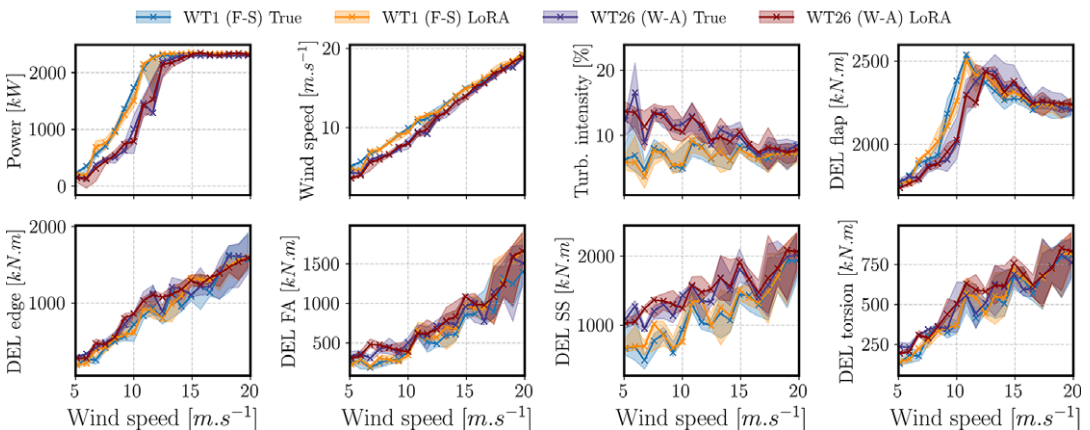


Figure 15. Power, wind speed, turbulence intensity, Flapwise DEL, Edgewise DEL, fore-aft (FA) DEL, side-to-side (SS) DEL and Torsional DEL curves for free-stream turbine (WT1; blue, true; orange, predicted using LoRA) and wake-affected turbine (WT26; purple, true; dark red, predicted using LoRA). SW–NW wind direction excluded. Shaded curves represent the 75% distribution percentiles.

do have better alignment with the HAWC2Farm ground truth. Here again, the model without pre-training predicts averaged out values over the entire farm.

Predicting wake effects. In this section, we focus on the performance over each variable for two different turbines. We present the results for the best LoRA model at 106 samples for all variables of two turbines of the Lillgrund wind farm: a free-flow wind turbine (WT1) and a wake-affected one (WT26). In order to ensure that WT1 is fully free-stream, we restrict the wind-rose by excluding SW–NW samples (see Figure 14). In Figure 15 we can observe the difference between the true and predicted power curves for the free-stream turbine (WT1, blue–orange) and the wake-affected one (WT26, purple–dark red), along with their relative position within the Lillgrund wind farm. In this figure it can be observed how the power curve for the free-stream turbine is much higher before rated power than for the wake affected turbine,

which also only achieves rated power at a higher wind speed due to the effect of turbulence. It is also recognisable that the LoRA GNN model is able to accurately model both power curves.

However, it should be noted in this figure that, for the free-flow turbine, the LoRA GNN model is over-predicting the power curve at rated power (wind speed above 11 m.s^{-1}). This has no physical meaning, as the turbine's power is restricted to its rated power and, therefore, the LoRA GNN model has not learned this important feature of the power curve. However, this can easily be corrected by applying a threshold. Additionally, in Figure 15, all output variables (true and predicted) are plotted for WT1 and WT26 against the global inflow wind speed. In this figure, the generalist GNN model is shown able to capture both free-stream (blue-orange) and wake-affected (purple-dark red) turbine behaviour for all variables. The model is quite capable of predicting the power curve, local wind speed and turbulence intensity. For other variables, namely some loads, like FA DEL, there are some discrepancies, but the overall trends are still being captured. We further observe that the model is able to clearly distinguish between free-stream turbines and wake-affected ones, further demonstrating that, also at a turbine-level, the fine-tuned GNN model is able to comprehend wake dynamics.

6.2.3. Negative transfer

A transfer learning approach is successful if it has the ability to adjust and exploit the inductive bias of a model trained on a source task such that it outperforms a model trained from scratch on the target task. When fine-tuning, a primary focus is to avoid negative transfer, a phenomenon that arises when the model's behavior, shaped by the optimization for the pre-training task, negatively impacts its performance on the new task. This issue often stems from the pre-existing weights and biases, initially optimized for the original tasks, not aligning with the new task's requirements. If this misalignment is not rectified during training, it can result in sub-optimal performance. In our case, some negative transfer is apparent in Figure 14, where the model predicts a power output beyond the rated value. To understand in more detail the negative transfer present in our fine-tuning framework, we plot in Figure 16 the predictive error of the LoRA and Vanilla models against the prediction errors for the model trained from scratch on only the HAWC2Farm data. For each model, 106 samples of the high-fidelity dataset were used for training, and results were gathered on the 200 samples of the test set. The color of the points indicate the inflow wind speed. We observe that both the LoRA and the Vanilla models mostly perform better than the Scratch model, with both distributions skewing below the parity line. For both models, strong cases of negative transfer seem to occur primarily for lower wind speeds. One potential explanation for poor performance

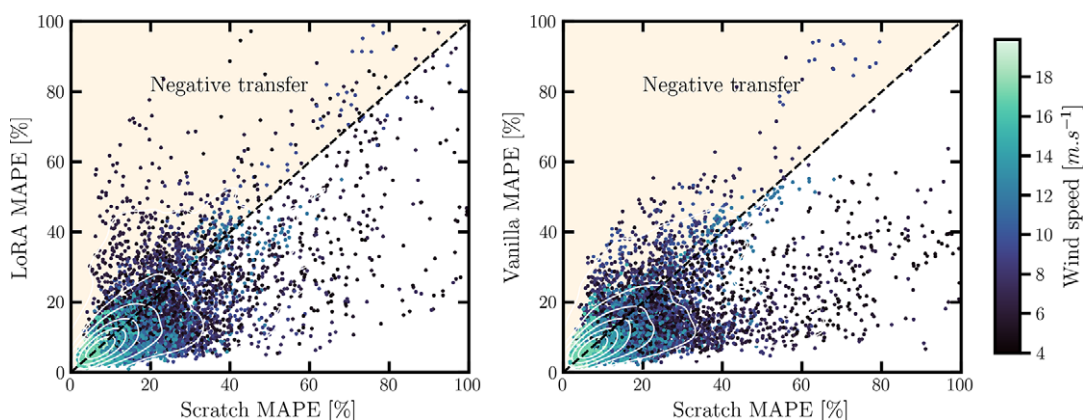


Figure 16. Side-by-side comparison of the distribution of LoRA and Vanilla MAPE results (computed by averaging MAPE over each output variable) plotted against the MAPE results of the Scratch model and colored by wind speed. Upper left parts of the plots are zones where negative transfer occurs, i.e. when pre-training harms model prediction for the downstream task. 106 training samples were used.

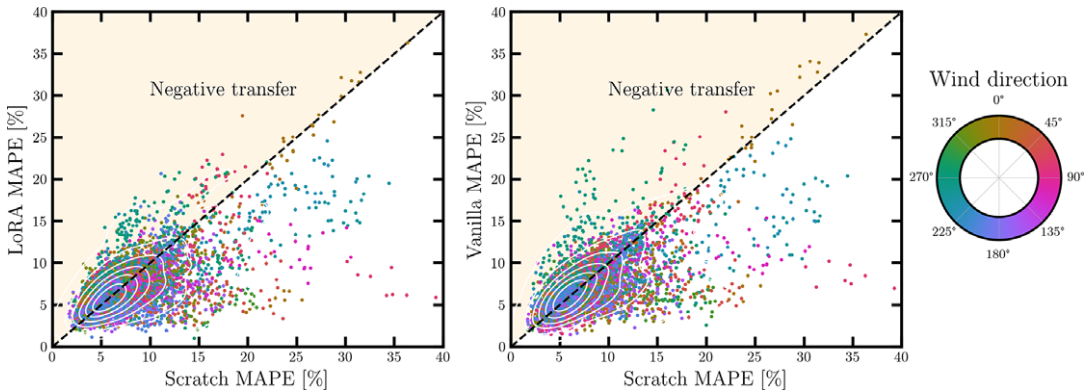


Figure 17. Side-by-side comparison of the distribution of LoRA and Vanilla MAPE results filtered for rated wind-speed only (computed by averaging MAPE over each output variable) plotted against the MAPE results of the Scratch model and colored by inflow direction. 106 training samples were used.

for low ambient wind speeds is the difficulty in accurately capturing the exponential behavior of the turbulence intensity when velocity is close to zero.

Excluding the low wind speed cases, we obtain Figure 17. In this case, we only show MAPE scores for wind speeds above rated power and color the points by inflow wind direction. From the plots we can identify particular inflow sectors which result in the worst cases of negative transfer: around 45°, 225° and 280° from North. In relation to the Lillgrund farm layout, these directions seem to correlate with the directions in which the strings of waked turbines are the longest. This is in line with our previous observation that PyWake has some difficulty modelling wake accumulation, and thus some negative legacy of the pre-training is observed.

6.3. Scaling up

In the previous section, we show that use of pre-trained models, even on data from low resolution flow simulators like PyWake, supports the GNN in learning basic wake physics and to understand the differences between free-stream and waked turbines. However, we further observe how intricate phenomena like boundary layer recuperation and wake effect accumulation over a farm are considerably more difficult to capture. Here, we attempt to improve on the GNN's expressivity by increasing the number of trainable parameters.

Our underlying argument behind this option lies with the fact that we are developing a generalist model, which should be able to adapt to any given layout, inflow condition, turbine type and even fidelity level. Therefore, it is reasonable to expend more time and resources on the training of the model, as its application further down the line would save much more time in computational terms. We can afford this because using LoRA to fine-tune the baseline model is rather fast, especially when only around 100 samples are used for fine-tuning. For our previous baseline model, we typically get single GPU training-step times of about 10s per epoch and a max VRAM footprint of about 1.2GB, which comfortably fits on the 24GB onboard VRAM of the RTX3090 GPU that we used. Thus, a much larger model can easily be trained and fine-tuned. We design a larger model with 8 message-passing layers and encoded dimensions of 1024 for all graph attributes, which yields approximately 40M trainable parameters. This model is first trained on the PyWake dataset and then fine-tuned using LoRA and compared here against the baseline LoRA model (1M trainable parameters), allowing us to understand the performance change given by scaling-up the model.

We first gather in Table 7 prediction error metrics for all variables for the LoRA 40M model, and compare them to the results for the best run of the LoRA 1M model, both trained on 106 simulations.

This table demonstrates that increasing the size of the model improves performance for nearly all outputs, by up to 9%, except for the edgewise blade DELs where performance is slightly degraded

Table 7. Performance (MAPE) for each estimated variable for the LoRA 1M and LoRA 40M models, fine-tuned on 106 HAWC2Farm samples, using the best performing data-splitting seed. For power, MAPE scores are only computed in cases when power is produced

Variable	LoRA 1M	LoRA 40M	Rel. diff. [%]
Power	4.46	4.06	−9.07
Wind speed	3.62	3.34	−7.75
Turbulence Intensity	9.06	8.72	−3.70
Flapwise DEL	1.99	1.84	−7.85
Edgewise DEL	15.10	15.34	1.57
FA DEL	11.38	10.35	−9.04
SS DEL	7.19	7.02	−2.47
Torsional DEL	6.98	6.77	−2.92

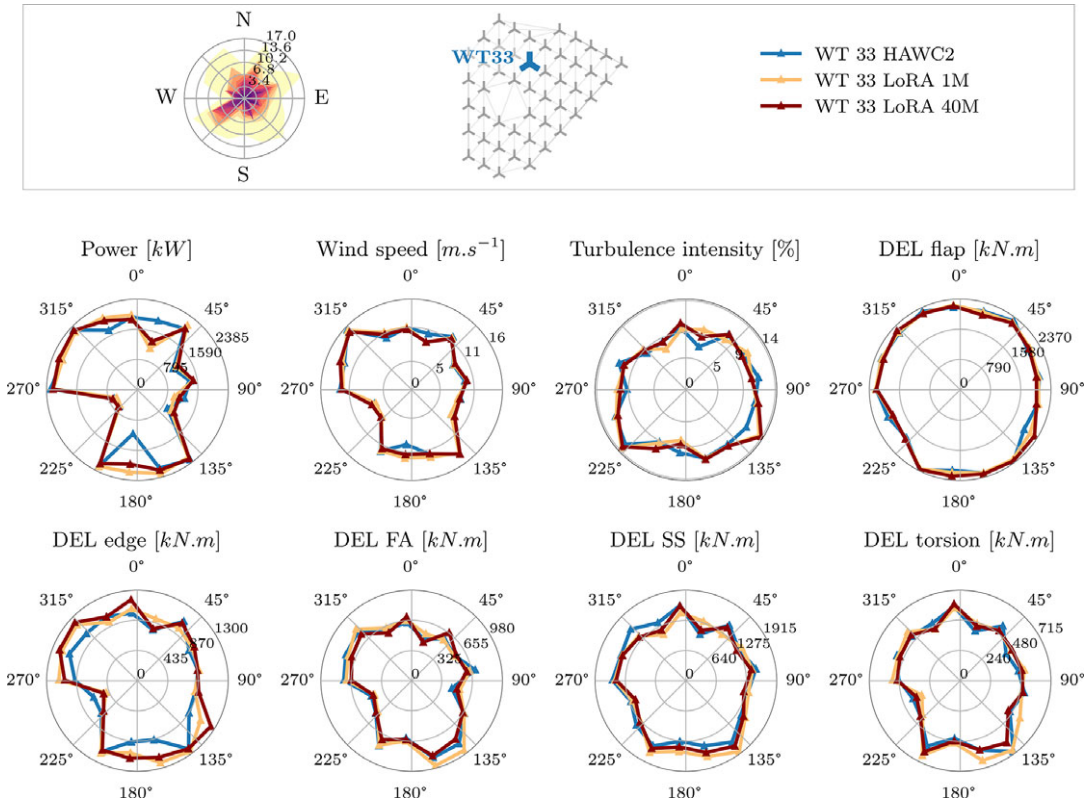


Figure 18. Polar plots of the predicted outputs averaged per wind direction for the LoRA 1M and LoRA 40M models trained on 106 HAWC2Farm simulations for a single center turbine of the Lillgrund farm.

although still comparable. For quantities like power, wind speed, fore-aft DEL, torsional DEL or flapwise DEL, the positive change is of note.

We then further investigate the LoRA 40M model performance by showing the polar plots comparison between both LoRA models and the HAWC2Farm ground truth per variable, given the 200 testing inflow samples (Figure 18). These are plotted for WT33 (waked) according to the incoming wind direction. This turbine is selected, as it is one of the most difficult locations to capture, given that it is located just behind

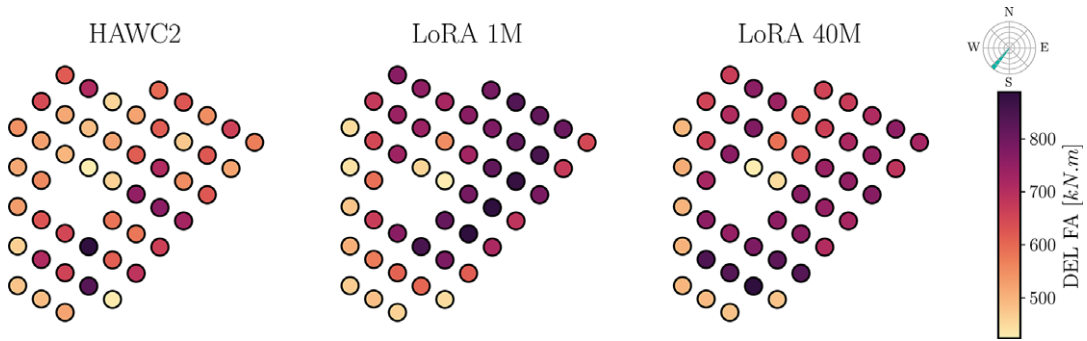


Figure 19. Comparison of the predicted fore-aft damage equivalent loads for the LoRA 1M and LoRA 40M models trained on 106 HAWC2Farm simulations for a single southwesterly inflow condition (9.4 m.s^{-1} and 9.9% turbulence intensity).

the 'gap' in the middle of the farm. In line with Table 7, the biggest improvements are seen for the fore-aft and torsional DELs, specifically around 135° . The highest errors occur for the edgewise DEL, with the largest over-prediction occurring for inflows around 180° . As previously mentioned, edgewise DELs are particularly difficult to capture due to their, in part, near-wake dependency and strongly non-linear behaviour.

We can finally spatially compare the farm-level behavior of the LoRA 1M, LoRA 40M and HAWC2 (ground truth), by plotting the farm predictions for each model in Figure 19. In this figure, we observe that an increased number of trainable parameters of the LoRA 40M model improves the spatial representativeness of the model. Specifically, immediately after the gap in the middle of the farm we notice how the original LoRA 1M model had an almost binary distinction for that region, with turbines in the vicinity of the gap in yellow, succeeded by an almost immediate over-prediction of loads. The overestimation phenomenon is transversal to the farm. As for LoRA 40M, despite some overestimation, it is better able to represent the nuance of fore-aft loading throughout the farm, especially for the back-row turbines and the overall overestimation is attenuated.

7. Discussion and conclusions

In this contribution, we present a predictive modelling approach at the level of wind turbine farms, seen as populations of structures that are represented by a graph. Specifically, we tackle the well-known Lillgrund offshore farm. To this end, we develop a 'generalist' graph neural network model first trained on relatively inexpensive numeric simulations from PyWake. We then fine-tune the model using a small amount of higher fidelity HAWC2Farm simulations of the target wind farm and turbine. This approach allows us to capitalize on the transfer learning capabilities of GNN, enabling training on inexpensive simulations before specializing the model for accurate prediction with higher fidelity data. Specifically, we compare four possible modeling strategies: training from scratch, fully re-training a pre-trained model (Vanilla), freezing all layers of the pre-trained model and just re-training the decoder ones, and employing low-rank adaptation (LoRA). Amongst these four approaches, we observe that LoRA and the Vanilla approach produce the best results, both in terms of overall percentage error, error per variable, and the capability of capturing spatial patterns (i.e., the load distribution over the farm). Both the LoRA and Vanilla models require lower amounts of data when fine-tuning than the Scratch and Decoder models to achieve the same accuracy levels. We furthermore observed that the inclusion of a pre-trained model significantly improves results, especially for layout pattern resolution when compared to training from scratch. Moreover, having access to the pre-trained model reduces the amount of data required to train a model in order to reach acceptable accuracy.

Thereafter, we focused on the physicality of the LoRA model for 106 training samples (half of the test samples), and observe a sufficiently precise prediction for all variables of interest (e.g., the power curve). We further analyze the proposed model's performance by testing for negative transfer and ensuring that it remains low. Finally, in order to increase spatial expressivity, we train a LoRA model with $\times 40$ more trainable parameters than the original LoRA model, with increased performance and better nuance on the farm-wide loading patterns.

This work introduces a powerful graph-based framework to fine-tune a flexible generalist model that can account for specific wind farm site layouts, different turbine types, and even diversified levels of simulation fidelity. This investigation further unveils current limitations. Complex wake behavior (boundary layer recuperation, near-wake effects, and wake effect accumulation) is better captured by the higher fidelity simulations. The goal of our transfer learning approach is to capture such complex physics while using low amounts high-fidelity data to do so, rather than simply training a surrogate model from scratch. In this contribution we demonstrate precisely this while, at the same time, establishing the limits of such an approach: not all intricacies of the more complex physics of the higher-fidelity simulations are captured by the model, despite the pre-training giving the model a good understanding of the basic physics (e.g., distinguishing between free-flow and wake-affected turbines). This is evident in the accumulation of wake effects throughout the farm and the corresponding influence of changes in the geometry (e.g., a gap in a row).

We attempt to circumvent this issue by increasing the pre-trained model's training parameters, with an increased performance and better nuance. However, we further notice that we are reaching the limits of the pre-training using the inexpensive flow simulator (PyWake) in terms of its ability to simulate wake physics. It is non-trivial to expect a model to learn intricate physical nuances not contained in its original pre-training dataset by simply fine-tuning. This, especially, using only 100 samples for the re-training of the model, while having to fine-tune for a new turbine type, with a different rotor size, which also impacts wake dynamics. A straightforward solution is to simply increase the number of mid- to high-fidelity samples used for fine-tuning or, even, pre-training a model on mid-fidelity simulators like HAWC2. Nonetheless, we can think of a more refined form of 'curriculum' training of this overarching GNN model in the future, one that is able to adapt to any new layout/turbine type, by dividing model pre-training into two stages. First, a stage of low-fidelity simulations (PyWake) can be used to train the GNN model and enable it to quickly learn fundamental (i.e., 'easy') physics like distinguishing between free-flow and wake affected turbines for a wide range of wind speeds, turbulence intensities, and different pseudo-layouts. In a second stage, the GNN model can be trained on mid-fidelity simulations (HAWC2), in a quantity and diversity sufficient for it to learn more intricate physical phenomena, not present in PyWake: the mechanics of boundary layer recuperation, near-wake effects, etc. We leave these refinements to be explored in future work.

The work presented in this contribution has the potential to be a powerful contributor for further advancements, such as accelerating layout design, optimizing wind farm instrumentation, and developing fatigue-aware control strategies. The introduced GNN framework, with its flexibility and low computational cost, could serve as a tractable surrogate that can be coupled with demanding optimization techniques like reinforcement learning. Furthermore, the automatic calculation of gradients by the GNN surrogate enables the use of gradient-based methods to develop new optimal control strategies, such as load-aware wake steering or curtailment. This synergy between GNN surrogates and optimization algorithms promises to enhance the overall performance and longevity of wind farms, balancing power generation with fatigue management.

Acknowledgements. We are grateful for the technical assistance of Pierre-Édouard Rethoré (DTU) with PyWake.

Author contribution. Conceptualization: G.D.; F.d.N.S.; I.A.; E.C. Methodology: G.D.; F.d.N.S.; I.A. Formal Analysis: G.D.; F.d.N.S. Data curation: G.D.; F.d.N.S.; I.A. Data visualisation: G.D.; F.d.N.S. Writing – Original draft: G.D.; F.d.N.S. Writing – Reviewing & Editing: I.A.; W.W.; E.C.; C.D. Funding Acquisition: E.C.; C.D. All authors approved the final submitted draft.

Competing interest. The authors declare no competing interests exist.

Data availability statement. We provide the associated code and pre-trained models at <https://github.com/gduthe/windfarm-gnn>. Furthermore, the dataset of PyWake simulations used for pre-training can be accessed in de N Santos et al. (2023a).

Funding statement. This work was supported by the BRIDGE Discovery Program of the Swiss National Science Foundation and Innosuisse (Grant No. 40B2-0_187087), as well as the French-Swiss project MISTERY funded by the French National Research Agency (ANR PRCI Grant No. 266157) and the Swiss National Science Foundation (Grant No. 200021L_212718). The authors would also like to acknowledge the Fonds Wetenschappelijk Onderzoek – Vlaanderen (FWO) for its long stay abroad grant (Grant No. V434422N).

Ethical standard. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

References

- Avenidaño-Valencia LD, Abdallah I and Chatzi E (2021) Virtual fatigue diagnostics of wake-affected wind turbine via gaussian process regression. *Renewable Energy*, 170, 539–561.
- Barber S et al. (2022) Development of a wireless, non-intrusive, mems-based pressure and acoustic measurement system for large-scale operating wind turbine blades. *Wind Energy Science*, 7(4), 1383–1398.
- Bastankhah M and Porté-Agel F (2014) A new analytical model for wind-turbine wakes. *Renewable energy*, 70, 116–123.
- Battaglia PW et al. (2018) Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.
- Bentsen LØ, Warakagoda ND, Stenbro R, and Engelstad P (2022) Wind park power prediction: Attention-based graph networks and deep learning to capture wake losses. *Journal of Physics: Conference Series* 2265, 022035.
- Black IM, Cevasco D, and Kolios A (2022) Deep neural network hard parameter multi-task learning for condition monitoring of an offshore wind turbine. *Journal of Physics: Conference Series* 2265, 032091.
- Bleeg J (2020) A graph neural network surrogate model for the prediction of turbine interaction loss. *Journal of Physics: Conference Series*, 1618, 062054.
- Bortolotti P, Tarres HC, Dykes KL, Merz K, Sethuraman L, Verelst D, and Zahle F (2019) *IEA Wind TCP Task 37: Systems engineering in wind energy-WP2. 1 Reference wind turbines*. Technical report, National Renewable Energy Lab (NREL), Golden, CO, United States.
- Breton SP, Sumner J, Sorensen JN, Hansen KS, Sarmast S and Ivanell S (2017) A survey of modelling methods for high-fidelity wind farm simulations using large eddy simulation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375, 20160097.
- Bronstein MM, Bruna J, LeCun Y, Szlam A and Vandergheynst P (2017) Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 18–42.
- Brown T, et al. (2020) Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33, 1877–1901.
- Bull L, et al. (2021) Foundations of population-based SHM, Part i: homogeneous populations and forms. *Mechanical Systems and Signal Processing*, 148, 107141.
- Casella G and Berger RL (2021) *Statistical inference*. Cengage Learning.
- Chang J (2023) github.com/ccntu/minLoRA <https://github.com/ccntu/minLoRA>.
- Chen Y-C (2017) A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology* 1, 161–187.
- Commission IE et al. (2005) Wind turbines-part 1: design requirements. *IEC 61400-1 Ed. 3*.
- Crespo A and Hernández J (1996) Turbulence characteristics in wind-turbine wakes. *Journal of Wind Engineering and Industrial Aerodynamics*, 61(1), 71–85.
- de N Santos F, Duthé G, Abdallah I, Réthoré P-É, Weijtjens W, Chatzi E, and Devriendt C (2024a). Multivariate prediction on wake-affected wind turbines using graph neural networks. In *EURODYN 2023. Proceedings of the XII International Conference on Structural Dynamics*: IOP Publishing <https://doi.org/10.3929/ethz-b-000674010>.
- de N Santos F, Duthé G, Abdallah I, Réthoré P-É, Weijtjens W, Chatzi E, and Devriendt C (2023a). Multivariate prediction on wake-affected wind turbines using graph neural networks (Eurodyn) database <https://doi.org/10.5281/zenodo.7760668>.
- de N Santos F, D’Antuono P, Robbelein K, Noppe N, Weijtjens W, and Devriendt C (2023b). Long-term fatigue estimation on offshore wind turbines interface loads through loss function physics-guided learning of neural networks. *Renewable Energy*.
- de N Santos F, Noppe N, Weijtjens W, and Devriendt C (2022) Data-driven farm-wide fatigue estimation on jacket-foundation OWTs for multiple SHM setups. *Wind Energy Science*, (1).
- de N Santos F, Noppe N, Weijtjens W, and Devriendt C (2024b) *Farm-wide interface fatigue loads estimation: a data-driven approach based on accelerometers*. Wind Energy.
- Dekking FM, Kraaikamp C, Lopuhaä HP, and Meester LE (2005) *A Modern Introduction to Probability and Statistics: Understanding Why and How*, 488. Springer.
- Delaunay B et al. (1934) Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7, 1–2.
- Dimitrov N, Natarajan A and Kelly M (2015) Model of wind shear conditional on turbulence and its impact on wind turbine loads. *Wind Energy*, 18(11), 1917–1931.
- Ding N, et al. (2023) Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence* 5, 220–235.

- Dodge J, Ilharco G, Schwartz R, Farhadi A, Hajishirzi H, and Smith N (2020) Fine-tuning pretrained language models: weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.
- Duthé G, Abdallah I, Barber S, and Chatzi E (2023a) Graph neural networks for aerodynamic flow reconstruction from sparse sensing. *arXiv preprint arXiv:2301.03228* <https://doi.org/10.48550/arXiv.2301.03228>.
- Duthé G, de N Santos F, Abdallah I, Réthoré P-E, Weijtjens W, Chatzi E, and Devriendt C (2023b). Local flow and loads estimation on wake-affected wind turbines using graph neural networks and pywake. *Journal of Physics: Conference Series*, 2505, 012014.
- Fey M and Lenssen JE (2019) Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.
- Fischereit J, Schaldemose Hansen K, Larsén XG., van der Laan MP, Réthoré P-E, and Murcia Leon JP (2022) Comparing and validating intra-farm and farm-to-farm wakes across different mesoscale and high-resolution wake models. *Wind Energy Science*, 7(3), 1069–1091.
- Gardner P, Bull L, Gosliga J, Dervilis N, and Worden K (2021) Foundations of population-based shm, part iii: Heterogeneous populations—mapping and transfer. *Mechanical Systems and Signal Processing*, 149, 107142.
- Gilmer J, Schoenholz SS, Riley PF, Vinyals O, and Dahl GE (2017) Neural message passing for quantum chemistry. In *International Conference on Machine Learning* (pp. 1263–1272): PMLR.
- Gosliga J, Gardner P, Bull L, Dervilis N, and Worden K (2021) Foundations of population-based SHM, part ii: heterogeneous populations—graphs, networks, and communities. *Mechanical Systems and Signal Processing*, 148, 107144.
- Guo Y, Shi H, Kumar A, Grauman K, Rosing T, and Feris R (2019) Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4805–4814.
- Halton JH (1960) On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2, 84–90.
- Hamilton W, Ying Z, and Leskovec J (2017) Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30.
- Hernández Q., Badías A, Chinesta F, and Cueto E (2022) Thermodynamics-informed graph neural networks. *arXiv preprint arXiv:2203.01874*.
- Hu EJ, et al. (2021) Lora: low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Hu W, Liu B, Gomes J, Zitnik M, Liang P, Pande V, and Leskovec J (2019) Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*.
- Hughes AJ, Poole J, Dervilis N, Gardner P, and Worden K (2023) Quantifying the value of information transfer in population-based shm. *arXiv preprint arXiv:2311.03083*.
- IEA (2023). World energy outlook 2023. <https://www.iea.org/reports/world-energy-outlook-2023>. License: CC BY 4.0 (report); CC BY NC SA 4.0 (Annex A).
- Janssens N and Meyers J (2023) Towards real-time optimal control of wind farms using large-eddy simulations. *Wind Energy Science Discussions*, 2023, 1–42.
- Johnson NL, Kotz S and Balakrishnan N (1995) *Continuous univariate distributions*, 289. John Wiley & Sons.
- Kingma DP and Ba J (2014) Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Lam R et al. (2023) Learning skillful medium-range global weather forecasting. *Science*, 382(6677), 1416–1421.
- Larsen GC, Madsen HA, Thomsen K and Larsen TJ (2008) Wake meandering: a pragmatic approach. *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, 11(4), 377–395.
- Larsen TJ and Hansen AM (2007) *How 2 HAWC2, The User's Manual*. Risø National Laboratory.
- Levick T, Neubert A, Friggo D, Downes P, Ruisi R and Bleeg J (2022) Validating the next generation of turbine interaction models. *Journal of Physics: Conference Series*, 2257, 012010.
- Li G, Xiong C, Thabet A and Ghanem B (2020a) Deepergcn: all you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.
- Li S, Zhang M, and Piggott M (2022) End-to-end wind turbine wake modelling with deep graph representation learning. *arXiv preprint arXiv:2211.13649*.
- Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A (2020b) Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*.
- Liew J, Andersen SJ, Trolldborg N and Göçmen T. (2022). Les verification of hawc2farm aeroelastic wind farm simulations with wake steering and load analysis. In *Journal of Physics: Conference Series*, volume 2265, 022069.
- Liew J, Riva R and Göçmen T (2023a) Efficient Mann turbulence generation for offshore wind farms with applications in fatigue load surrogate modelling. *Journal of Physics: Conference Series* 2626, 012050.
- Liew JY, Riva R and Göçmen T (2023b) Simulated fatigue load data set of the Lillgrund wind farm. <https://doi.org/10.11583/DTU.21399645.v1> https://data.dtu.dk/articles/dataset/Simulated_fatigue_load_data_set_of_the_Lillgrund_wind_farm/21399645.
- Long M, Wang J, Ding G, Sun J and Yu PS (2013) Transfer feature learning with joint distribution adaptation. *Proceedings of the IEEE international conference on computer vision* (pp. 2200–2207).
- Loshchilov I and Hutter F (2016) Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.
- Madsen HA, Larsen TJ, Pirrung GR, Li A and Zahle F (2020) Implementation of the blade element momentum model on a polar grid and its aeroelastic load impact. *Wind Energy Science*, 5(1), 1–27.
- Micheli A. (2009). Neural network for graphs: a contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3), 498–511.

- Musin OR** (1997) Properties of the delaunay triangulation. In *Proceedings of the Thirteenth Annual symposium on Computational geometry* (pp. 424–426).
- Mylonas C., Abdallah I, and Chatzi E** (2021a) Conditional variational autoencoders for probabilistic wind turbine blade fatigue estimation using supervisory, control, and data acquisition data. *Wind Energy*, 24(10), 1122–1139.
- Mylonas C, Abdallah I and Chatzi E** (2021b) Relational VAE: a continuous latent variable model for graph structured data. *arXiv preprint arXiv:2106.16049* <https://arxiv.org/abs/2106.16049>.
- Niayifar A and Porté-Agel F** (2016) Analytical modeling of wind farms: a new approach for power prediction. *Energies*, 9(9), 741.
- OpenFAST** (2016). github.com/openfast/openfast <https://github.com/OpenFAST/openfast>.
- O'Neill ME** (2014) *Pcg: a family of simple fast space-efficient statistically good algorithms for random number generation*. ACM Transactions on Mathematical Software.
- Pan SJ, Tsang IW, Kwok JT and Yang Q** (2010). Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2), 199–210.
- Park J and Park J** (2019) Physics-induced graph neural network: An application to wind-farm power estimation. *Energy*, 187, 115883.
- Paszke A, et al.** (2019). Pytorch: an imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32, 8024–8035). Curran Associates, Inc.
- Pedersen MM, van der Laan P, Friis-Møller M, Rinker J and Réthoré P-E** (2019). Dtuwindenergy/pywake: Pywake. *Zenodo [code]*, 10.
- Pfaff T, Fortunato M, Sanchez-Gonzalez A and Battaglia PW** (2020) Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*.
- Porté-Agel F, Bastankhah M, and Shamsoddin S** (2020) Wind-turbine and wind-farm flows: a review. *Boundary-Layer Meteorology*, 174(1), 1–59.
- Radford A, Narasimhan K, Salimans T, Sutskever I, et al.** (2018). Improving language understanding by generative pre-training.
- Rinker JM, Soto Sagredo E and Bergami L** (2021) The importance of wake meandering on wind turbine fatigue loads in wake. *Energies*, 14(21), 7313.
- Saltelli A, Annoni P, Azzini I, Campolongo F, Ratto M and Tarantola S** (2010) Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer physics communications*, 181(2), 259–270.
- Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, and Battaglia P** (2020) Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning* (pp. 8459–8468): PMLR.
- Sanchez-Gonzalez A, Heess N, Springenberg JT, Merel J, Riedmiller M, Hadsell R and Battaglia P** (2018). Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning* (pp. 4470–4479): PMLR.
- Scarselli F, Gori M, Tsoi AC, Hagenbuchner M and Monfardini G** (2008) The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61–80.
- Schøler J, Riva R, Andersen S, Leon J M, van der Laan M, Risco JC. and Réthoré P-E** (2023) Rans-ad based ann surrogate model for wind turbine wake deficits. *Journal of Physics: Conference Series*, 2505, 012022.
- Simpson T, Dervilis N and Chatzi E** (2021) Machine learning approach to model order reduction of nonlinear systems via autoencoder and lstm networks. *Journal of Engineering Mechanics*, 147(10), 04021061.
- Sobol' IM** (1967) On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4), 784–802.
- Sood I, Simon E, Vitsas A, Blockmans B, Larsen GC and Meyers J** (2022) Comparison of large eddy simulations against measurements from the lillgrund offshore wind farm. *Wind Energy Science*, 7(6), 2469–2489.
- Tan C, Sun F, Kong T, Zhang W, Yang C, and Liu C** (2018) A survey on deep transfer learning. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III* 27 (pp. 270–279): Springer.
- Tsialiamanis G, Mylonas C, Chatzi EN, Wagg DJ, Dervilis N and Worden K** (2022) On an application of graph neural networks in population-based shm. *Data Science in Engineering*, 9, 47–63 .
- Veers P, et al.** (2022) Grand challenges in the design, manufacture, and operation of future wind turbine systems. *Wind Energy Science Discussions* 1–102.
- Veers P, et al.** (2019) Grand challenges in the science of wind energy. *Science*, 366(6464), eaau2027.
- Veličković P** (2023). Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79, 102538.
- Veličković P, Cucurull G, Casanova A, Romero A, Lio P, and Bengio, Y.** (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- Welling M and Kipf TN** (2016) Semi-supervised classification with graph convolutional networks. *Journal International Conference on Learning Representations (ICLR 2017)*.
- Wilson B, Wakes S, and Mayo M** (2017) Surrogate modeling a computational fluid dynamics-based wind turbine wake simulation using machine learning. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–8): IEEE.
- Worden K, et al.** (2020) A brief introduction to recent developments in population-based structural health monitoring. *Frontiers in Built Environment*, 6, 146.
- Xu K, Hu W, Leskovec J, and Jegelka S** (2018) How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.
- Xu K, Zhang M, Li J, Du SS, Kawarabayashi K-i, and Jegelka S** (2020) How neural networks extrapolate: from feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*.

- Yin H, Ou Z, Fu J, Cai Y, Chen S, and Meng A** (2021) A novel transfer learning approach for wind power prediction based on a serio-parallel deep learning architecture. *Energy*, 234, 121271.
- Yu M, et al.** (2020). Superposition graph neural network for offshore wind power prediction. *Future Generation Computer Systems*, 113, 145–157.
- Zraggen J, Ulmer M, Jarlskog E, Pizza G, and Huber LG** (2021) Transfer learning approaches for wind turbine fault detection using deep learning. In *PHM Society European Conference*, volume 6 (pp. 12–12).